Creating and Adding Custom IP

2021.2

Objectives

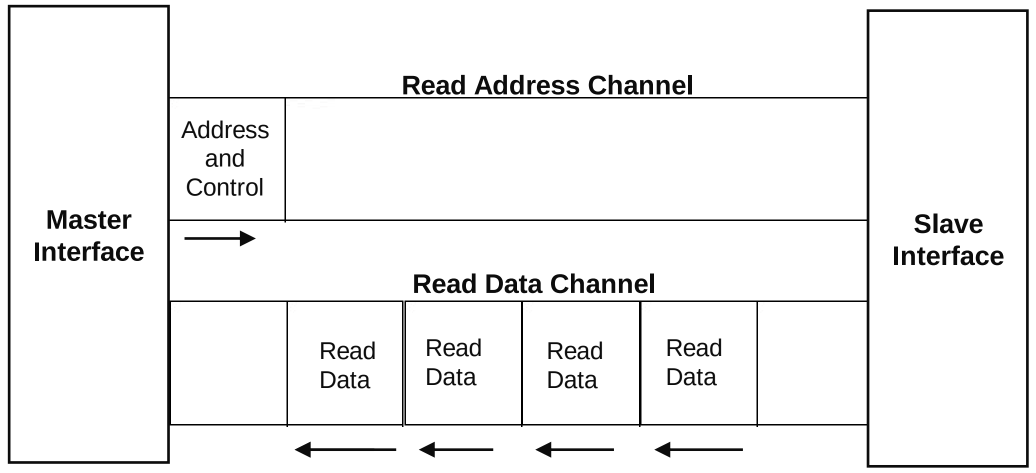
- **After completing this module, you will be able to:**
  - Describe the AXI4 transactions
  - Summarize the AXI4 valid/ready acknowledgment model
  - Discuss the AXI4 transactional modes of overlap and simultaneous operations
  - Describe the operation of the AXI4 streaming protocol
  - List the steps involved in creating and packaging IP

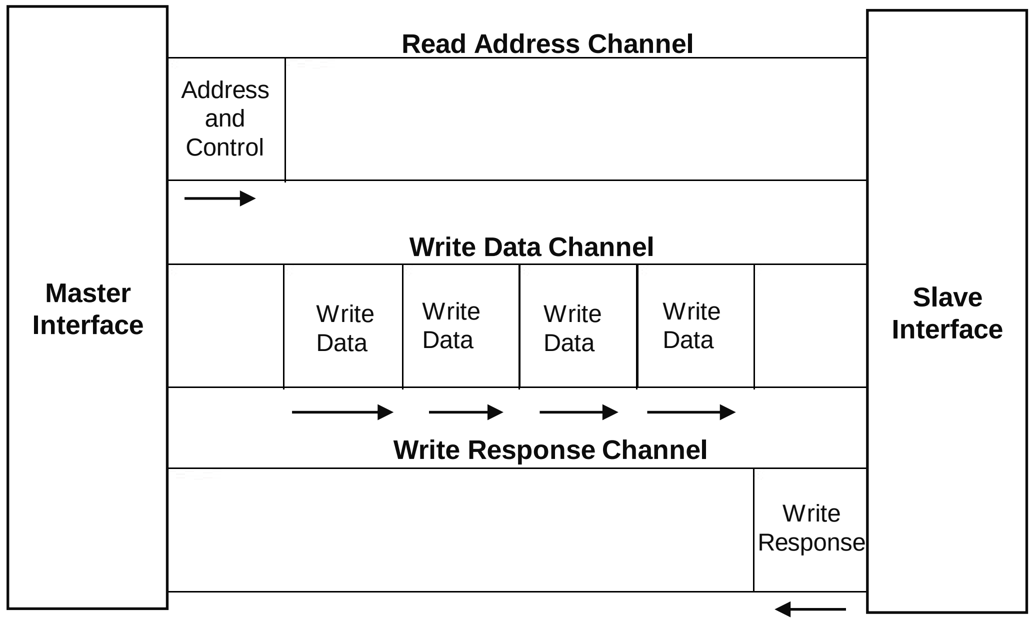
Outline

- ***AXI4 Transactions***
  - AXI4-Lite Slave
  - AXI4-Lite Master
  - AXI4 Slave/Master
- Create and Package IP
- Custom IP
- Summary

Basic AXI Transaction Channels

- Read address channel
- Read data channel
- Write address channel
- Write data channel
- Write response channel
  - Non-posted write model
    - There will always be a "write response"





All AXI Channels Use A Basic “VALID/READY” Handshake

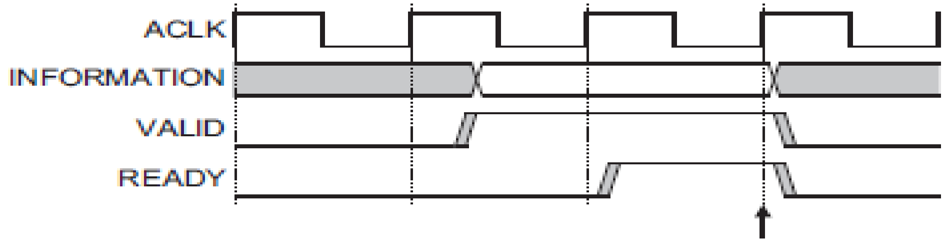
- *SOURCE* asserts and holds VALID when DATA is available
- *DESTINATION* asserts READY if able to accept DATA

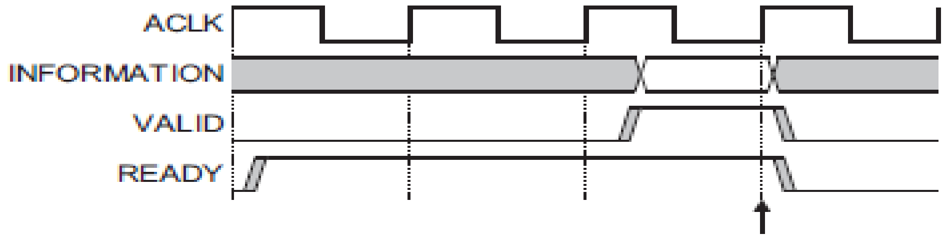
- DATA transferred when VALID and READY = 1

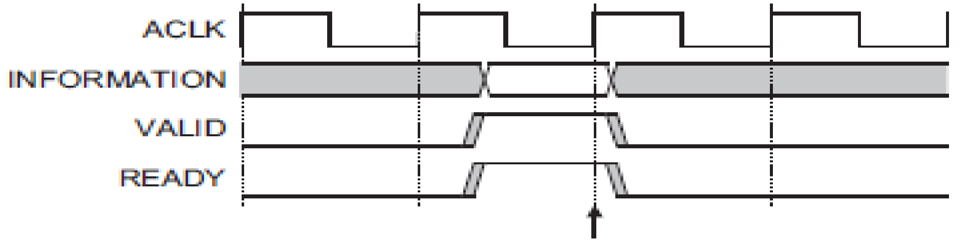
- *SOURCE* sends next DATA (if an actual data channel) or deasserts VALID
- *DESTINATION* deasserts READY if no longer able to accept DATA

AXI Interface: Handshaking

- AXI uses a valid/ready handshake acknowledge
- Each channel has its own valid/ready
  - Address (read/write)
  - Data (read/write)
  - Response (write only)
- Flexible signaling functionality
  - Inserting wait states
  - Always ready
  - Same cycle acknowledge

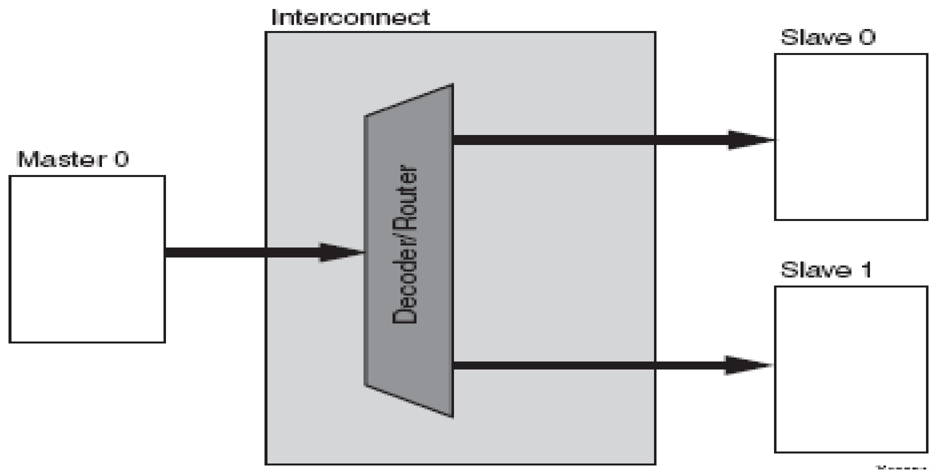


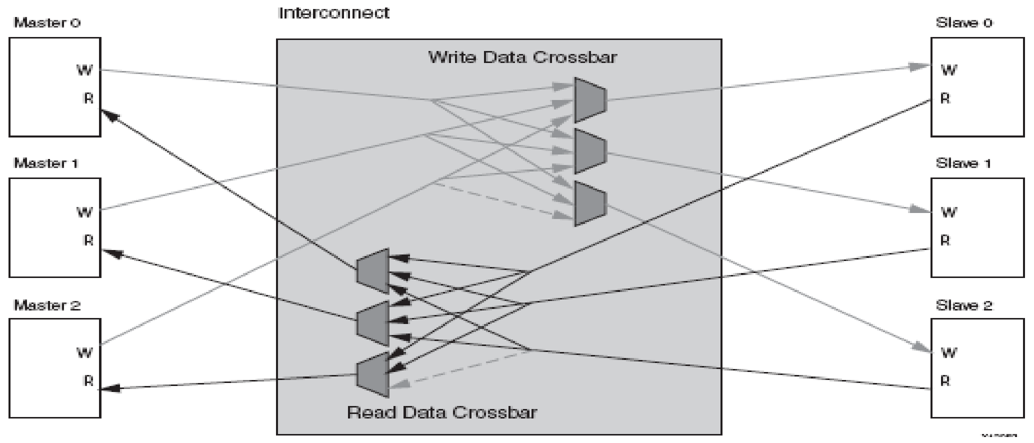




AXI Interconnect

- axi_interconnect component
  - Highly configurable
    - Pass Through
    - Conversion Only
    - N-to-1 Interconnect
    - 1-to-N Interconnect
    - N-to-M Interconnect – full crossbar
    - N-to-M Interconnect – shared bus structure
- Decoupled master and slave interfaces

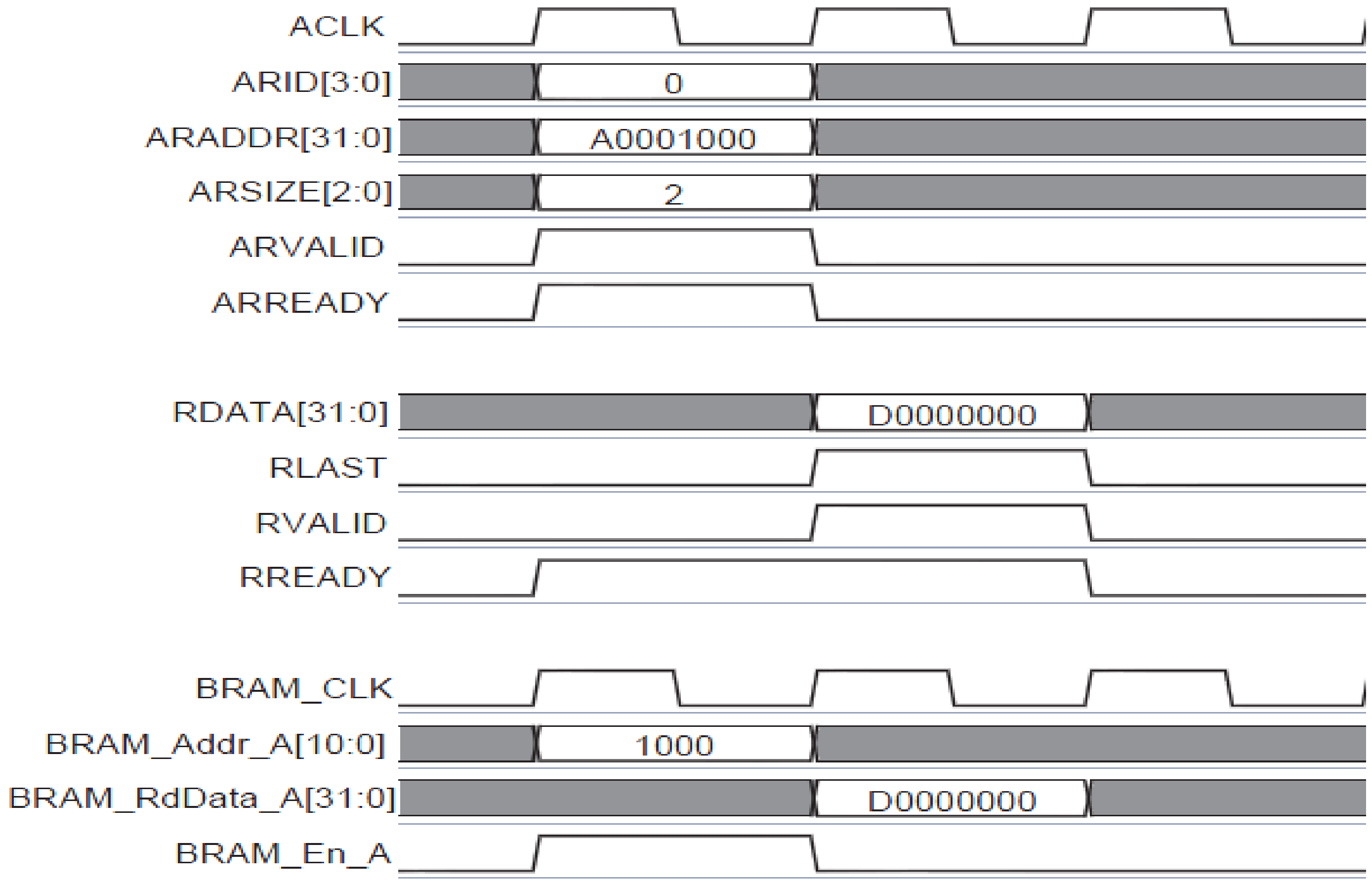




AXI4 Transactions: AXI4 Slave/Master

AXI4

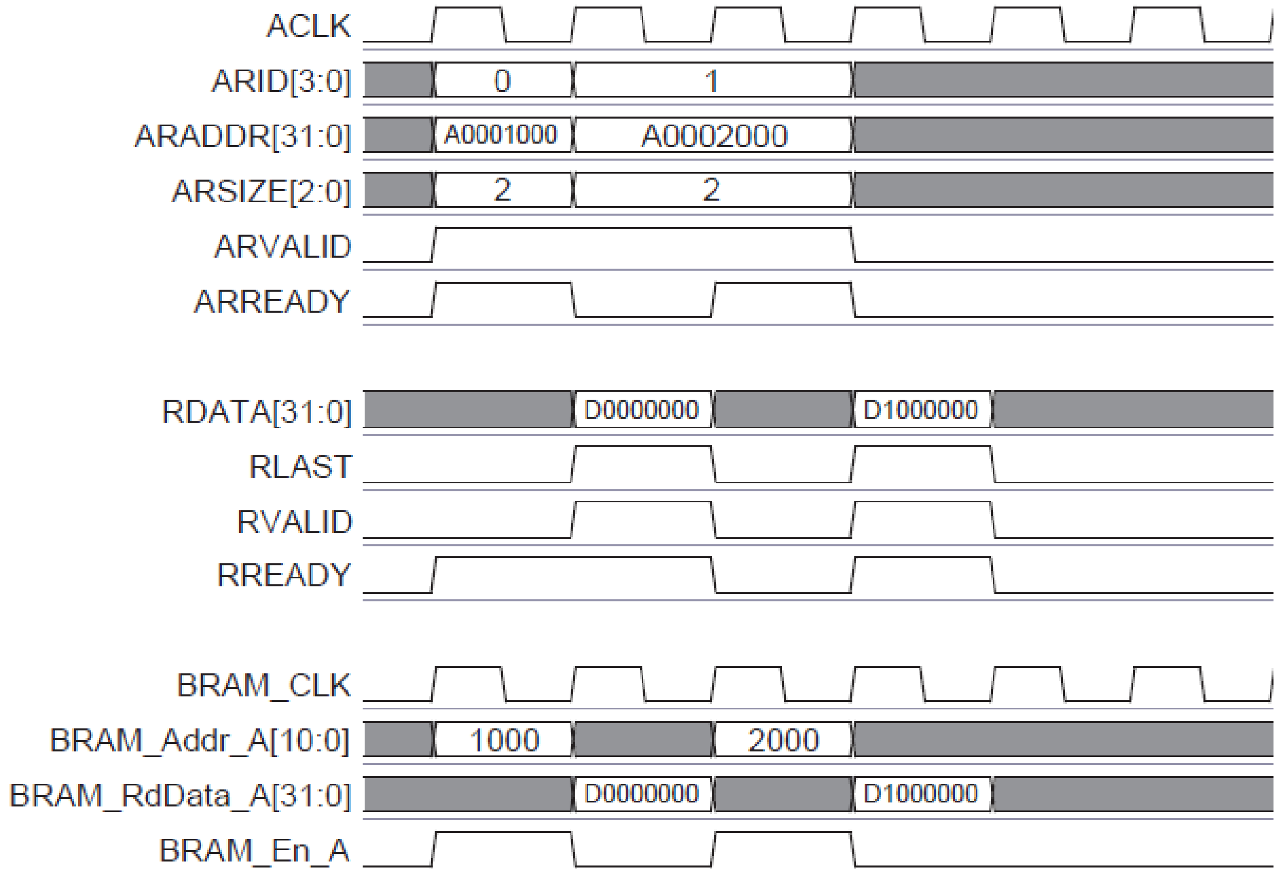
- Superset of AXI4-Lite
- Extra signals to support additional features
  - Burst transactions
  - Transaction ordering
- Not all AXI4 signals used by Xilinx IP
  - Ignored, or Pass through
- Single Read/Write transactions similar to AXI lite
  - Transaction ID
  - LAST signal



Example of read from AXI4 BRAM

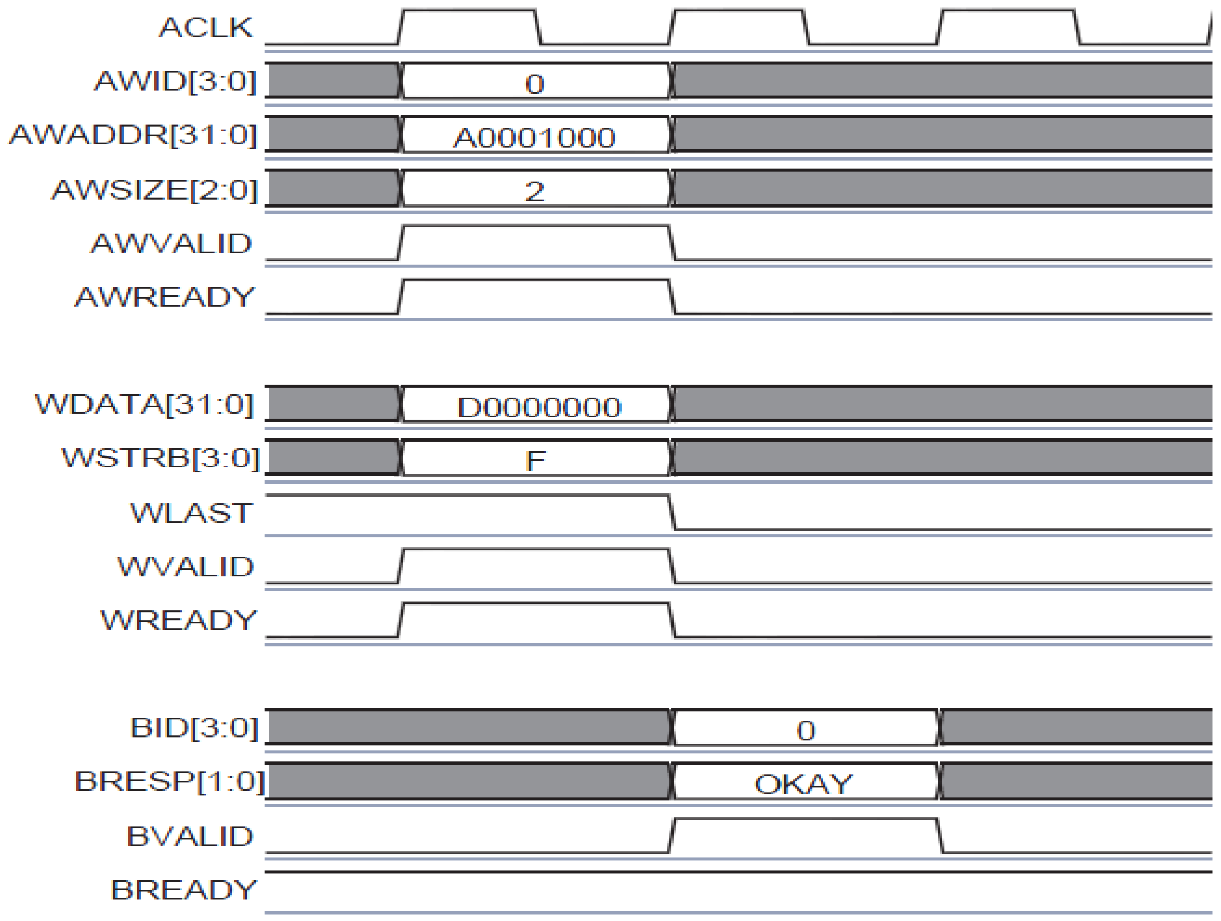
AXI4 Read Transaction
Multiple reads, not burst transaction

- Multiple read transactions are identified by different read IDs
  - For a burst transaction only one read ID would have been used
- Separate RLAST for corresponding read transactions
- ARSIZE=2 indicates 4 bytes are being read
  - 0 : 1 byte
  - 1 : 2 bytes
  - 2 : 4 bytes



Example of multiple reads from AXI4 BRAM

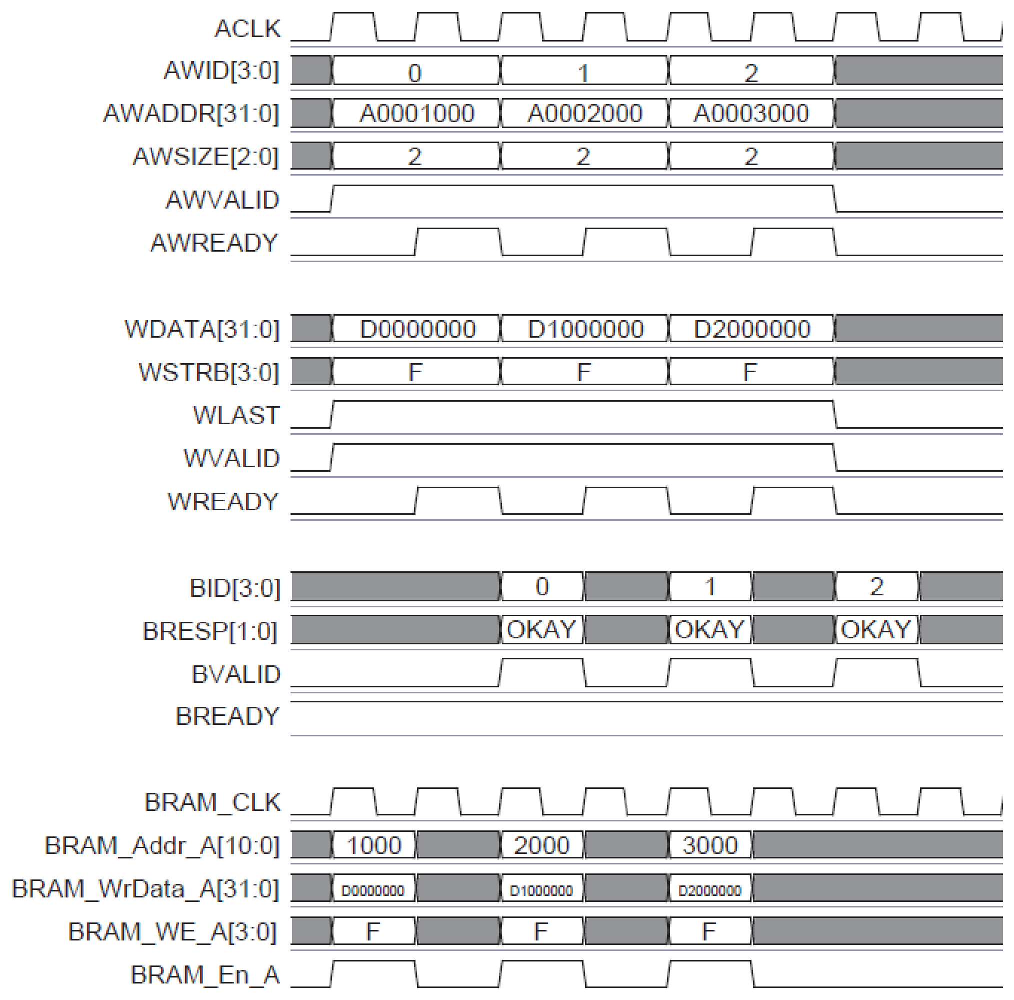
AXI4 Write Transaction



Example of AXI4 write transaction

AXI4 Write Transaction Multiple write, not burst

- Multiple write transactions are identified by different write IDs (AWID)
  - For a burst transaction only one write ID would have been used
- Separate OKAY status for corresponding write transactions
- WSTRB=0xF indicates entire word is being written



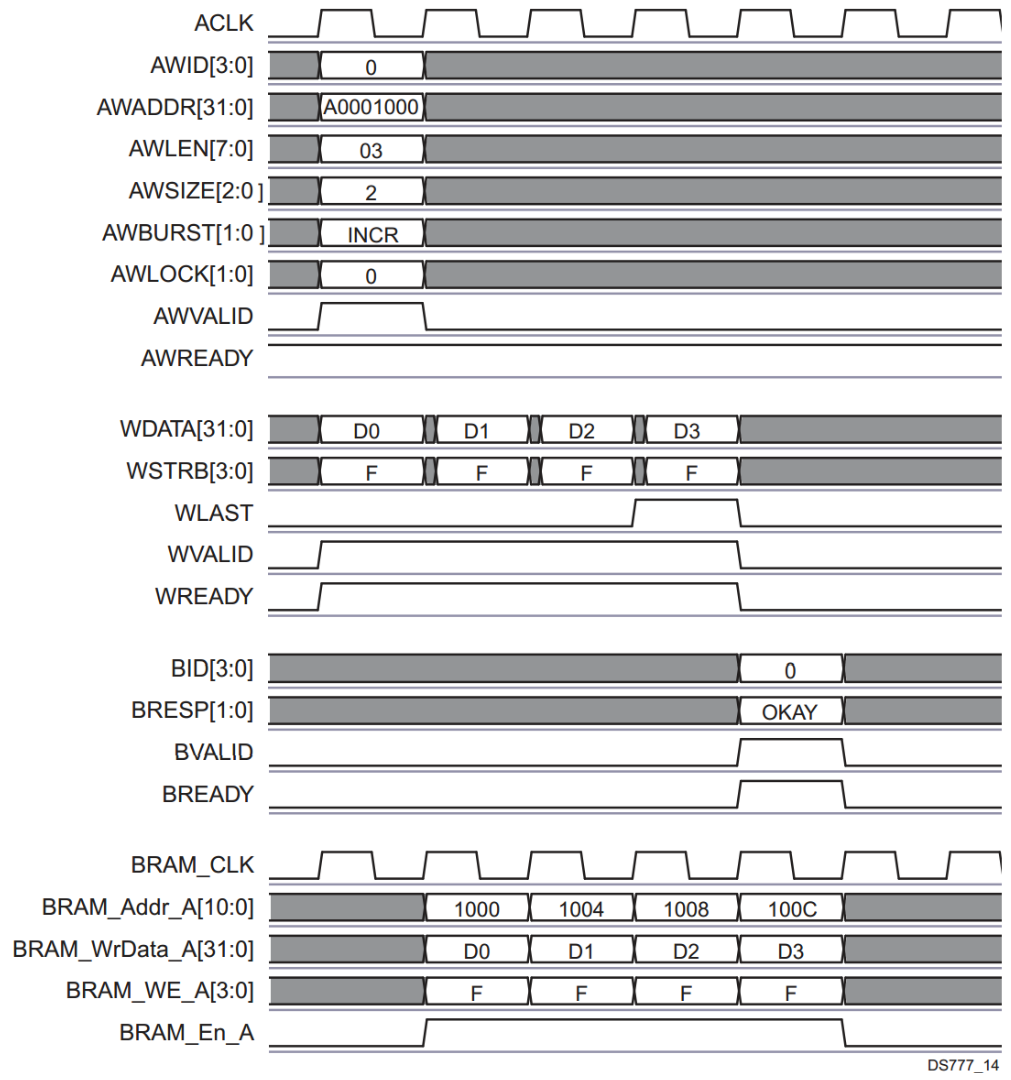
Example of AXI4 multiple write transactions

AMD XILINX

AXI4 Write Burst

Burst Data Phase 4 Writes

- Single Burst transaction
  - AWLEN = 3 (Burst Size)
  - Length = AWLEN+1 (4 transactions)
- INCR from Base AWADDR
- WSTRB=0xF indicates entire word is being written
- LAST indicates end of burst
  - AWLEN also indicates the size of the burst
- A burst must not cross a 4KB Boundary



Example of AXI4 write burst transaction

Documentation

- Xilinx AXI Reference Guide, UG761
  - AXI Usage in Xilinx FPGAs
    - Introduce key concepts of the AXI protocol
    - Explains what features of AXI Xilinx has adopted
- ARM specifications
  - AMBA AXI Protocol Version 2.0
  - AMBA 4 AXI4-Stream Protocol Version 1.0
  - http://infocenter.arm.com/help/topic/com.arm.doc.set.amba

AXI4 Signals (AXI4, AXI4-Lite)

	AXI4	AXI4-Lite
Glbl	ACLK	ACLK
	ARESETN	ARESETN
Write Address	AWID	
	AWADDR	AWADDR
	AWLEN	
	AWSIZE	
	AWBURST	
	AWLOCK	
	AWCACHE	AWCACHE
	AWPROT	AWPROT
	AWQOS	
	AWREGION	
	AWUSER	
	AWVALID	AWVALID
	AWREADY	AWREADY

	AXI4	AXI4-Lite
Write Data	WDATA	WDATA(1)
	WSTRB	WSTRB(2)
	WLAST	
	WUSER	
	WVALID	WVALID
	WREADY	WREADY
Write Response	BID	
	BRESP	BRESP(3)
	BUSER	
	BVALID	BVALID
	BREADY	BREADY

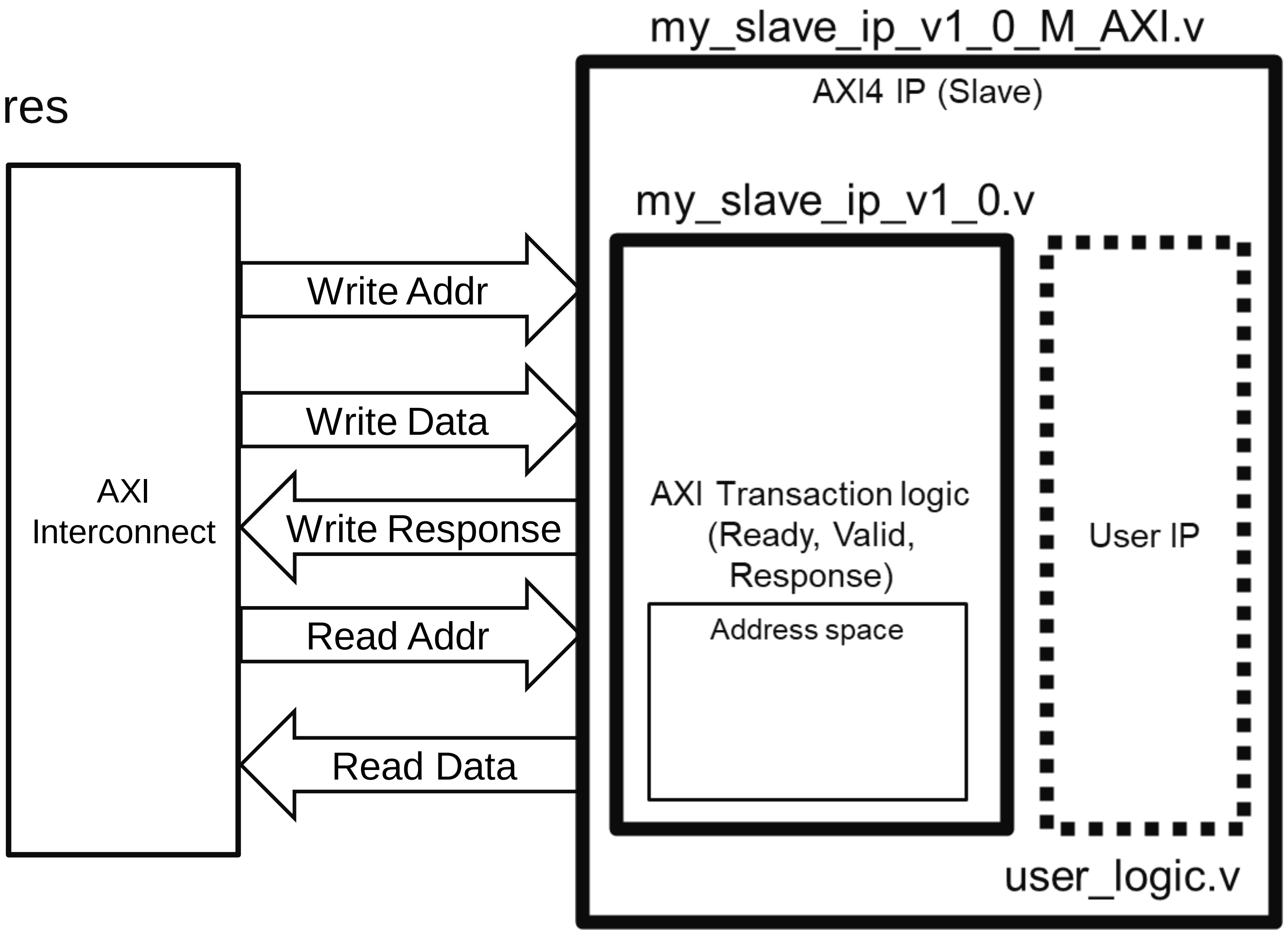
	AXI4	AXI4-Lite
Read Address	ARID	
	ARADDR	ARADDR
	ARLEN	
	ARSIZE	
	ARBURST	
	ARLOCK	
	ARCACHE	ARCACHE
	ARPROT	ARPROT
	ARQOS	
	ARREGION	
	ARUSER	
	ARVALID	ARVALID
	ARREADY	ARREADY

	AXI4	AXI4-Lite
Read Data	RID	
	RDATA	RDATA(1)
	RRESP	RRESP(3)
	RLAST	
	RUSER	
	RVALID	RVALID
	RREADY	RREADY

(1) 32-bit only
(2) Slave can ignore assuming all bytes valid
(3) EXOKAY not supported

AXI4

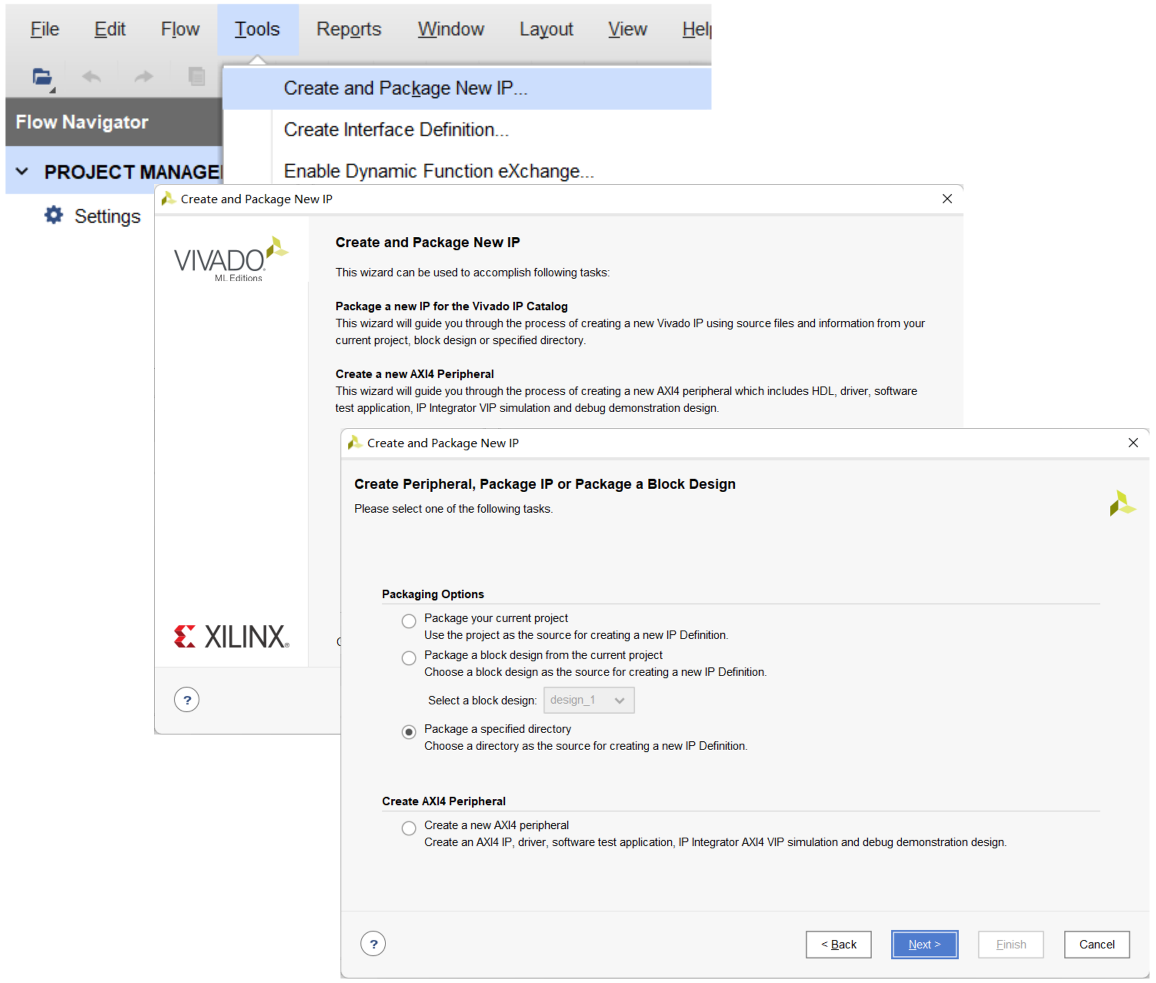
- Extra signals to support additional features
  - Superset of AXI4-Lite
  - Burst transactions
  - Transaction ordering
- Not all AXI4 signals used by Xilinx IP
  - Ignored, or Pass through
- Single Read/Write transactions
  - Similar to AXI lite
  - Transaction ID
  - LAST signal
- Memory space



Create and Package IP: Create IP

Create and Package IP

- The Create and Package IP wizard allows you to
  - *Create* a template for a new peripheral
  - *Package* the current project
    - Only available if project is open
  - *Package* an existing project
- The packager allows IP to be included in the IP Catalog for distribution
- It uses IP-XACT format
- Complete set of files included
  - Source code, Constraints, Test Benches (simulation files), documentation



Create and Package New IP

Create and Package New IP

This wizard can be used to accomplish following tasks:

Package a new IP for the Vivado IP Catalog
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project, block design or specified directory.

Create a new AXI4 Peripheral
This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator VIP simulation and debug demonstration design.

Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

Please select one of the following tasks.

Packaging Options

- Package your current project
  Use the project as the source for creating a new IP Definition.
- Package a block design from the current project
  Choose a block design as the source for creating a new IP Definition.
  Select a block design: design_1
- [✓ Package a specified directory]
  Choose a directory as the source for creating a new IP Definition.

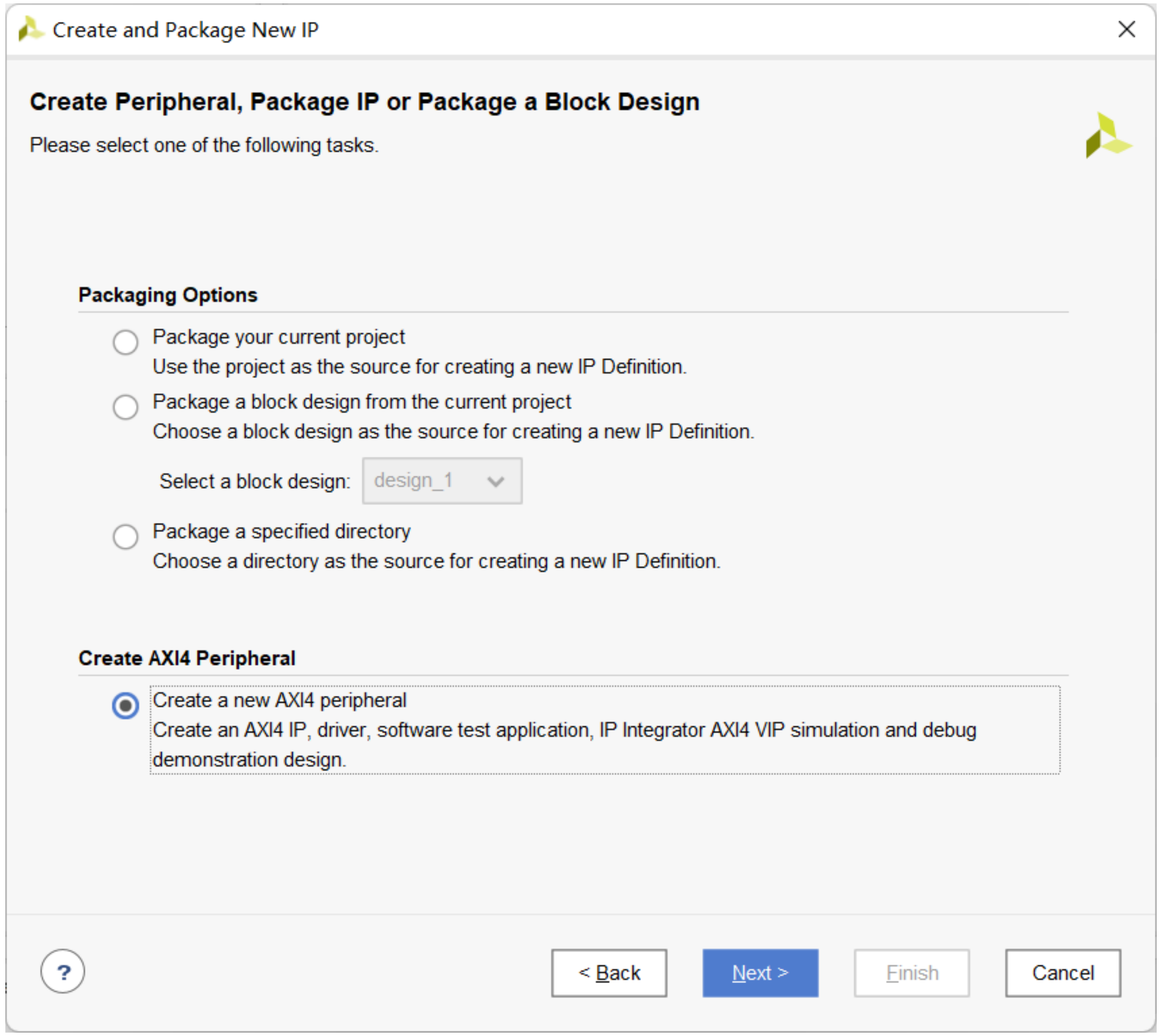
Create AXI4 Peripheral

- Create a new AXI4 peripheral
  Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

AMD XILINX

Create Custom IP

- Create and Package IP Wizard
- Generates HDL template for
  - Slave/Master
    - AXI Lite/Full/Stream
- Optionally Generates
  - Software Driver
    - Only for AXI Lite and Full slave interface
  - Test Software Application
  - AXI4 BFM Example



Create Peripheral, Package IP or Package a Block Design

Please select one of the following tasks.

Packaging Options

( ) Package your current project
Use the project as the source for creating a new IP Definition.

( ) Package a block design from the current project
Choose a block design as the source for creating a new IP Definition.

Select a block design: design_1

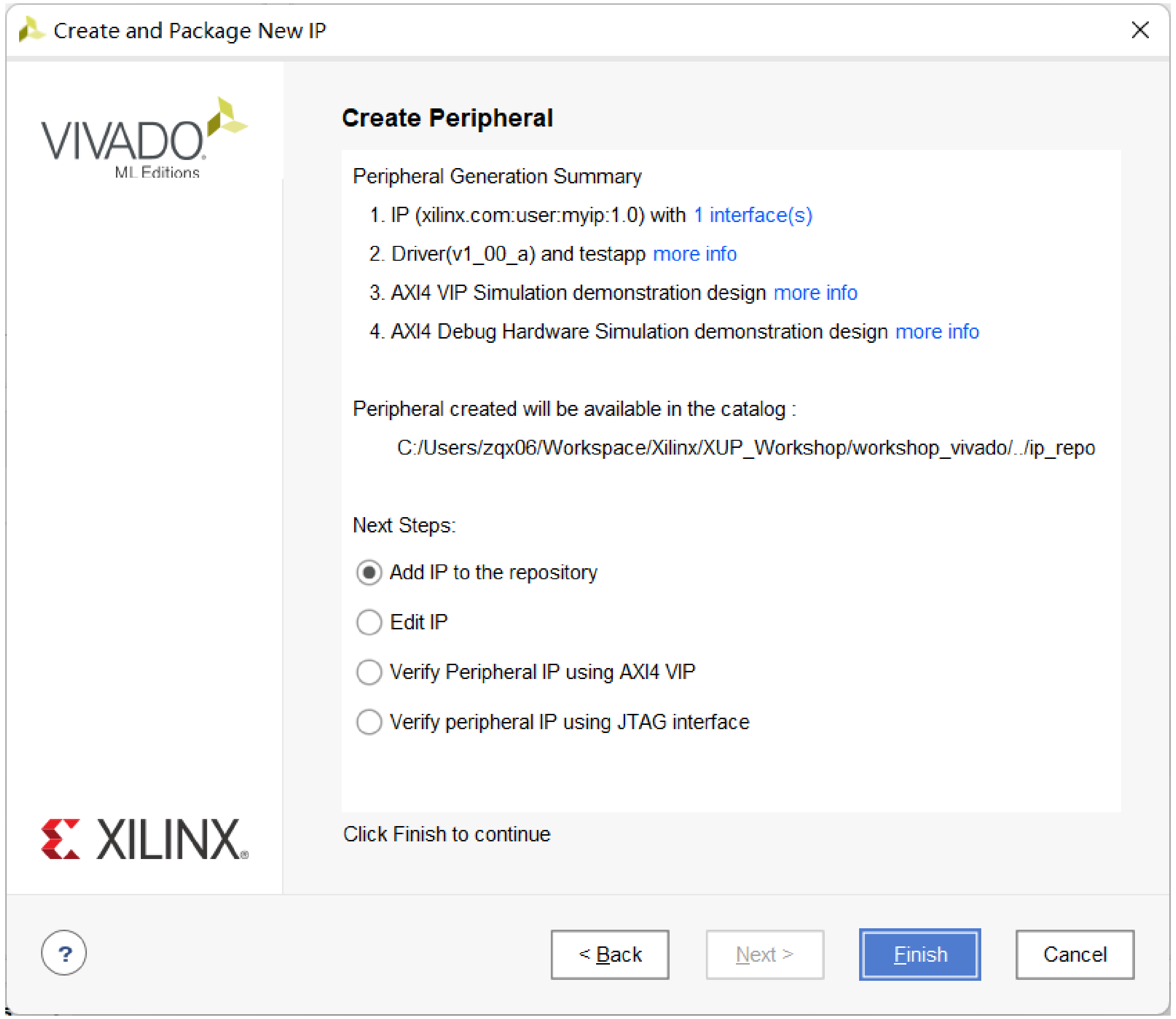
( ) Package a specified directory
Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

[✓ Create a new AXI4 peripheral]
Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

Create Custom IP

- Add IP to repository
- Edit IP
- Verify
  - BFM Simulation Example Design
  - JTAG



Create Peripheral

Peripheral Generation Summary

1. IP (xilinx.com:user:myip:1.0) with 1 interface(s)
2. Driver(v1_00_a) and testapp more info
3. AXI4 VIP Simulation demonstration design more info
4. AXI4 Debug Hardware Simulation demonstration design more info

Peripheral created will be available in the catalog :

C:/Users/zqx06/Workspace/Xilinx/XUP_Workshop/workshop_vivado/../ip_repo

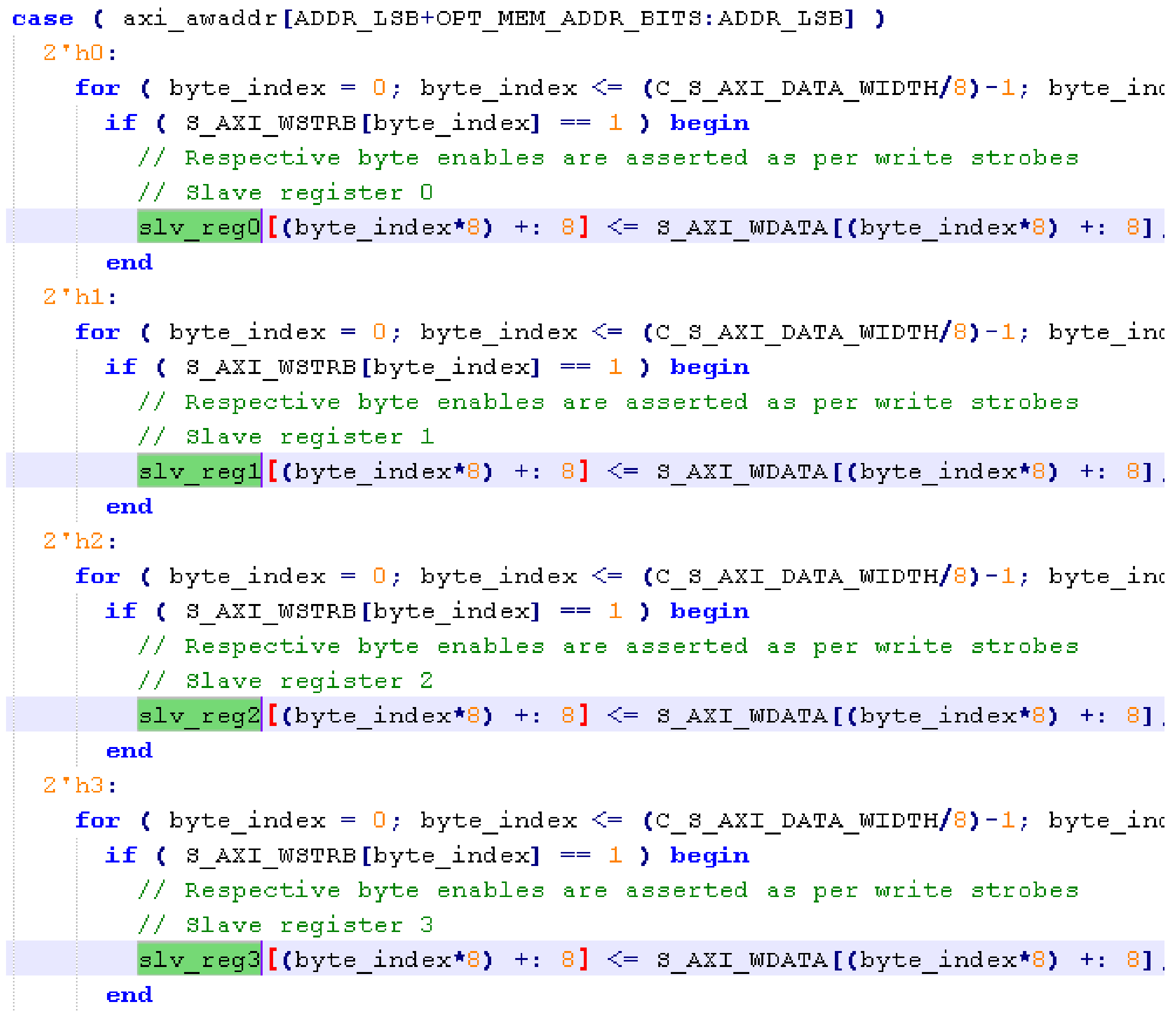
Next Steps:

- [✓ Add IP to the repository]
- Edit IP
- Verify Peripheral IP using AXI4 VIP
- Verify peripheral IP using JTAG interface

Click Finish to continue

Generated Template for AXI Lite

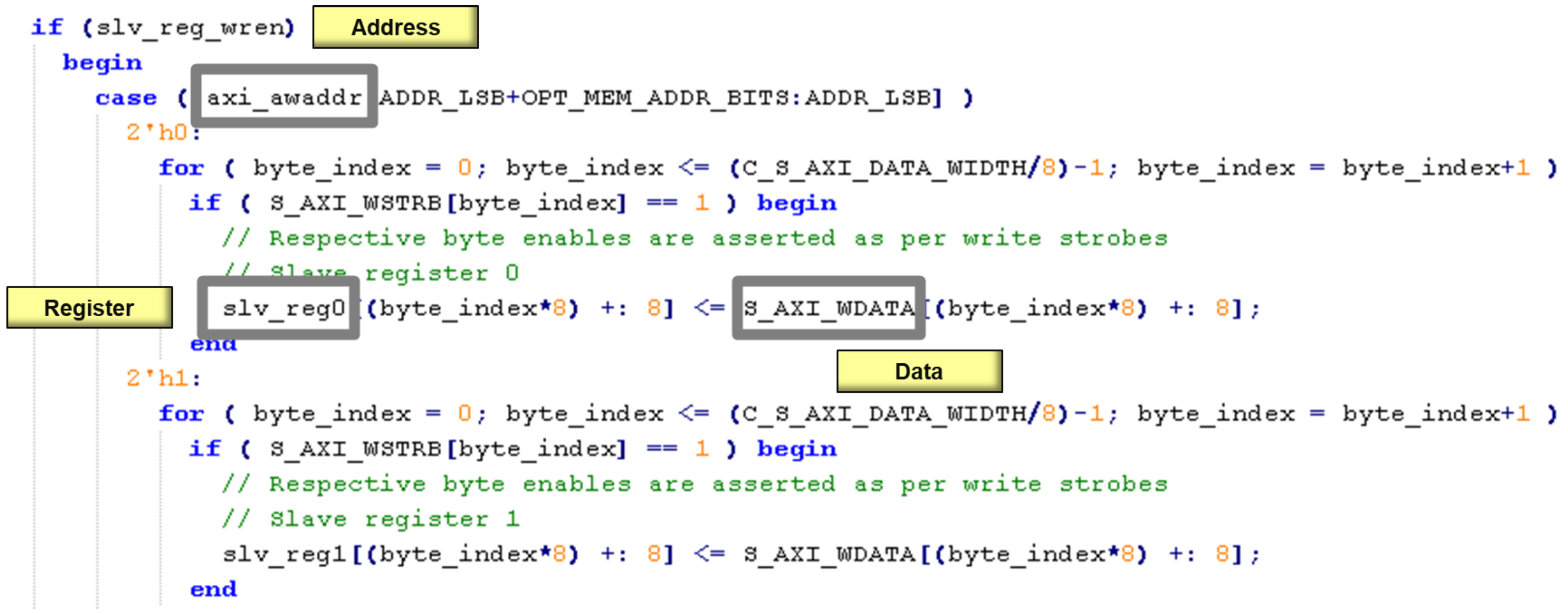
- HDL implementation of AXI Interface
  - 32-bit data width
- User specifies required number of registers (minimum 4)
- Read/write to/from Registers implemented
- User logic can be easily connected
- User logic can be a hierarchical design



```
case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
  2'h0:
    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_in
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as per write strobes
        // Slave register 0
        slv_reg0[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8]
      end
  2'h1:
    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_in
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as per write strobes
        // Slave register 1
        slv_reg1[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8]
      end
  2'h2:
    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_in
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as per write strobes
        // Slave register 2
        slv_reg2[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8]
      end
  2'h3:
    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_in
      if ( S_AXI_WSTRB[byte_index] == 1 ) begin
        // Respective byte enables are asserted as per write strobes
        // Slave register 3
        slv_reg3[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8]
      end
```

HDL AXI Lite

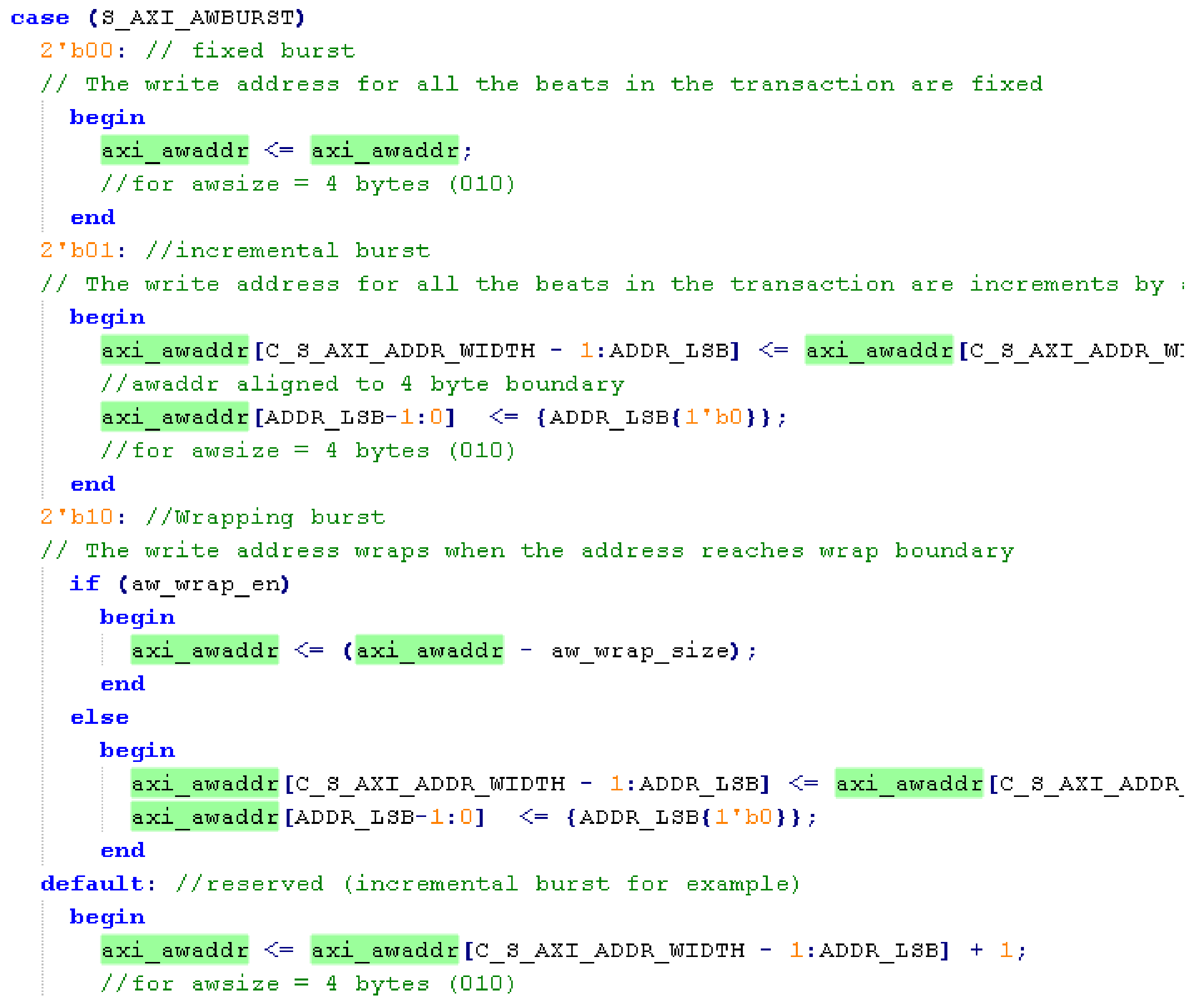
- Connect user logic to registers, or modify design



```
if (slv_reg_wren)
  begin
    case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
      2'h0:
        for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
          if ( S_AXI_WSTRB[byte_index] == 1 ) begin
            // Respective byte enables are asserted as per write strobes
            // Slave register 0
            slv_reg0[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
          end
      2'h1:
        for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
          if ( S_AXI_WSTRB[byte_index] == 1 ) begin
            // Respective byte enables are asserted as per write strobes
            // Slave register 1
            slv_reg1[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
          end
```

Generated Template for AXI Full

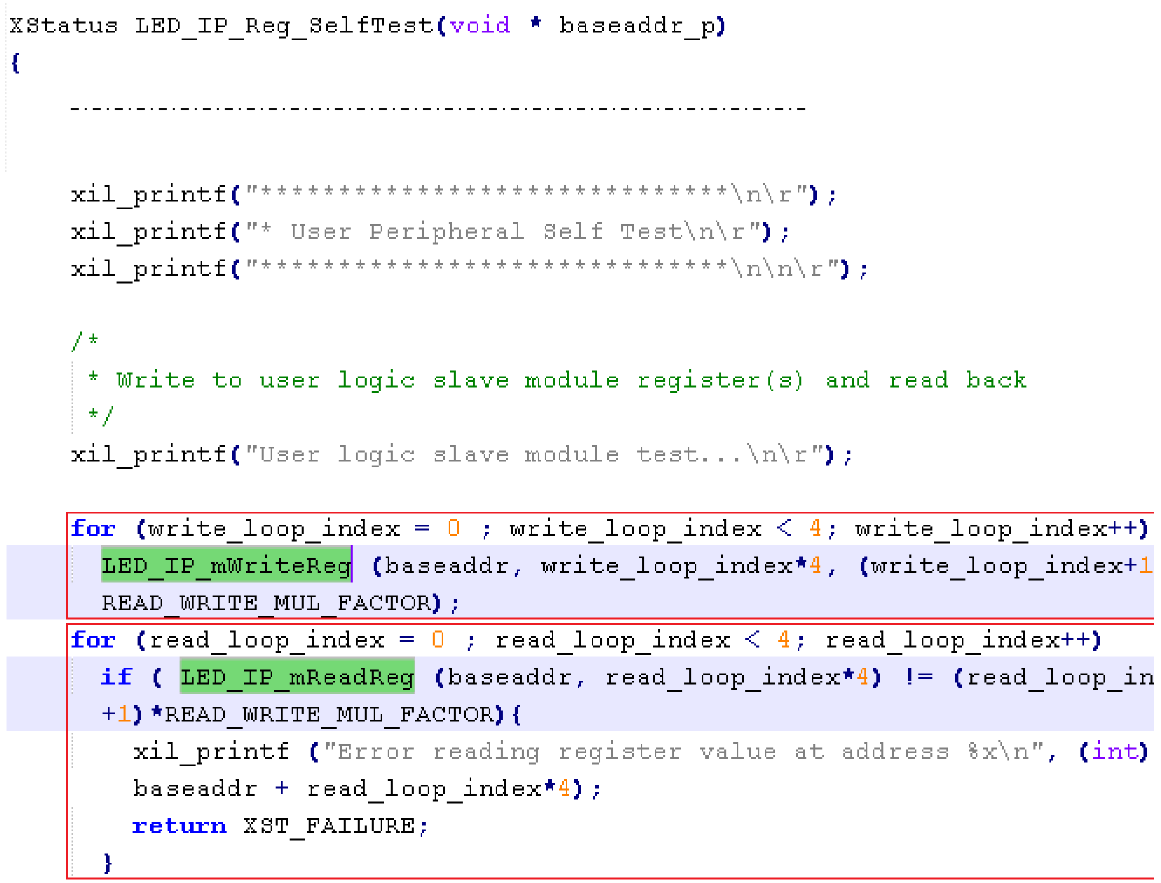
- HDL AXI Full Interface
  - 32-bit data interface
- Burst transaction support implemented
  - Specify size of memory space
  - Up to 1024 Bytes
- Example code implementing block memory
  - User logic can connect or replace this section



```
case (S_AXI_AWBURST)
  2'b00: // fixed burst
  // The write address for all the beats in the transaction are fixed
    begin
      axi_awaddr <= axi_awaddr;
      //for awsize = 4 bytes (010)
    end
  2'b01: //incremental burst
  // The write address for all the beats in the transaction are increments by
    begin
      axi_awaddr[C_S_AXI_ADDR_WIDTH - 1:ADDR_LSB] <= axi_awaddr[C_S_AXI_ADDR_W
      //awaddr aligned to 4 byte boundary
      axi_awaddr[ADDR_LSB-1:0]  <= {ADDR_LSB{1'b0}};
      //for awsize = 4 bytes (010)
    end
  2'b10: //Wrapping burst
  // The write address wraps when the address reaches wrap boundary
    if (aw_wrap_en)
      begin
        axi_awaddr <= (axi_awaddr - aw_wrap_size);
      end
    else
      begin
        axi_awaddr[C_S_AXI_ADDR_WIDTH - 1:ADDR_LSB] <= axi_awaddr[C_S_AXI_ADDR_
        axi_awaddr[ADDR_LSB-1:0]  <= {ADDR_LSB{1'b0}};
      end
  default: //reserved (incremental burst for example)
    begin
      axi_awaddr <= axi_awaddr[C_S_AXI_ADDR_WIDTH - 1:ADDR_LSB] + 1;
      //for awsize = 4 bytes (010)
```

Files created

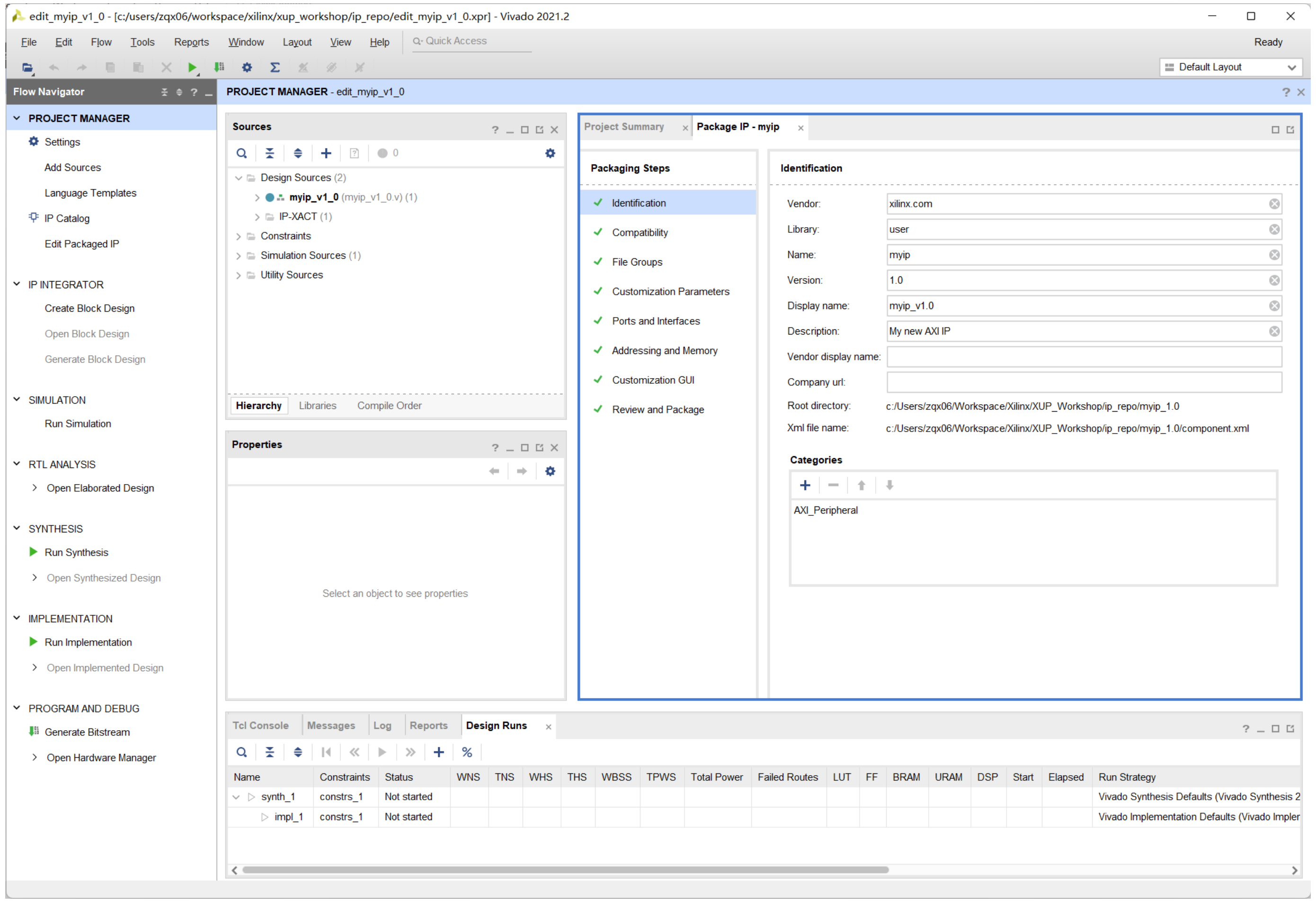
- component.xml
  - IP XACT description
- bd
  - Block Diagram tcl file
- drivers
  - SDK and software files (c code)
  - Simple register/memory read/write functionality
  - Simple SelfTest() code
- hdl
  - Verilog/VHDL (should be inside project directory
- xgui
  - GUI tcl file



```
XStatus LED_IP_Reg_SelfTest(void * baseaddr_p)
{
    -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-


    xil_printf("******************************\n\r");
    xil_printf("* User Peripheral Self Test\n\r");
    xil_printf("******************************\n\n\r");

    /*
     * Write to user logic slave module register(s) and read back
     */
    xil_printf("User logic slave module test...\n\r");

    for (write_loop_index = 0 ; write_loop_index < 4; write_loop_index++)
      LED_IP_mWriteReg (baseaddr, write_loop_index*4, (write_loop_index+1
      READ_WRITE_MUL_FACTOR);
    for (read_loop_index = 0 ; read_loop_index < 4; read_loop_index++)
      if ( LED_IP_mReadReg (baseaddr, read_loop_index*4) != (read_loop_in
      +1)*READ_WRITE_MUL_FACTOR){
        xil_printf ("Error reading register value at address %x\n", (int)
        baseaddr + read_loop_index*4);
        return XST_FAILURE;
      }
```

Edit IP

- New Vivado project will open
- Template files have been generated and are added to the project
- IP can now be edited
  - Modify existing template files, add user source files
- IP Packager will be open
  - Last step is to Package IP



Identification

Vendor:	xilinx.com
Library:	user
Name:	myip
Version:	1.0
Display name:	myip_v1.0
Description:	My new AXI IP
Vendor display name:	
Company url:	
Root directory:	c:/Users/zqx06/Workspace/Xilinx/XUP_Workshop/ip_repo/myip_1.0
Xml file name:	c:/Users/zqx06/Workspace/Xilinx/XUP_Workshop/ip_repo/myip_1.0/component.xml

Categories

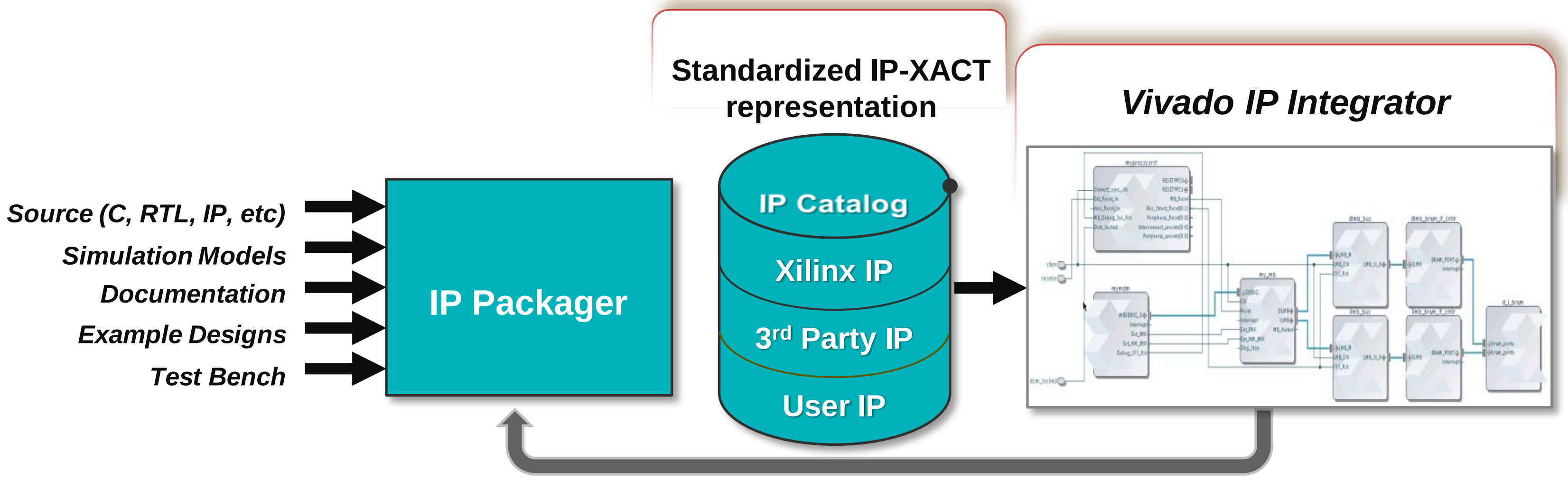
AXI_Peripheral

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	Not started																Vivado Synthesis Defaults (Vivado Synthesis 2
impl_1	constrs_1	Not started																Vivado Implementation Defaults (Vivado Impler

Create and Package IP: Package IP

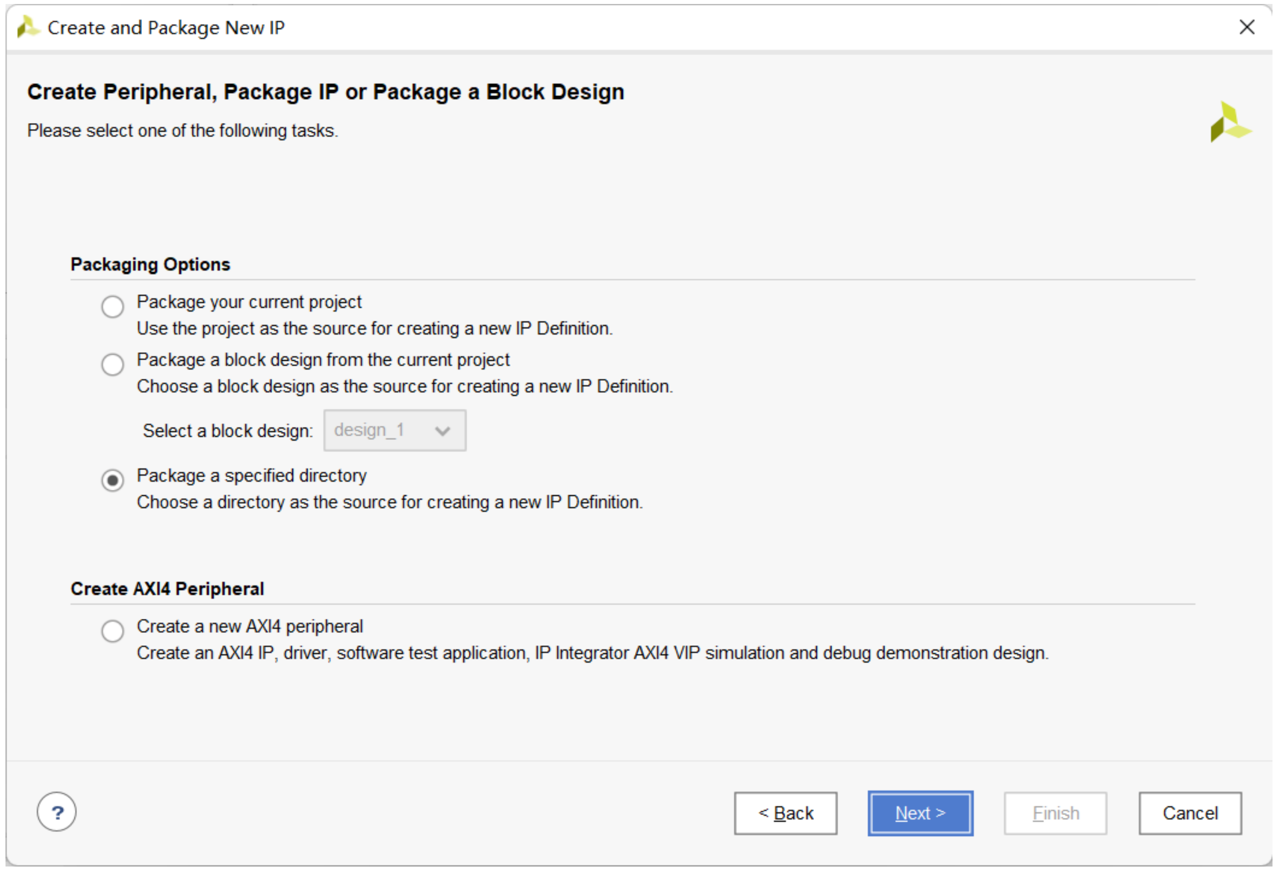
Reusing Your IP

- IP from many sources can be packaged and made available in Vivado
- All IP available in the Vivado IP Catalog can be used to create IP Integrator designs
- Any IP Integrator diagram can be quickly packaged as a single complex IP



Package IP

- Package current project
  - Must be open!
  - Package generated HDL
- Package a directory (another project/source files)
  - Option to package as a library core
  - Core can be referenced by other IP
  - Core not available standalone (Will not display in IP Catalog)
- Create New AXI4 Peripheral (previous section)
  - Will also need to be packaged
  - Similar steps to package for all three flows


Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

Please select one of the following tasks.

Packaging Options

( ) Package your current project
Use the project as the source for creating a new IP Definition.

( ) Package a block design from the current project
Choose a block design as the source for creating a new IP Definition.

Select a block design: design_1

(● Package a specified directory)
Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

( ) Create a new AXI4 peripheral
Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

IP-XACT

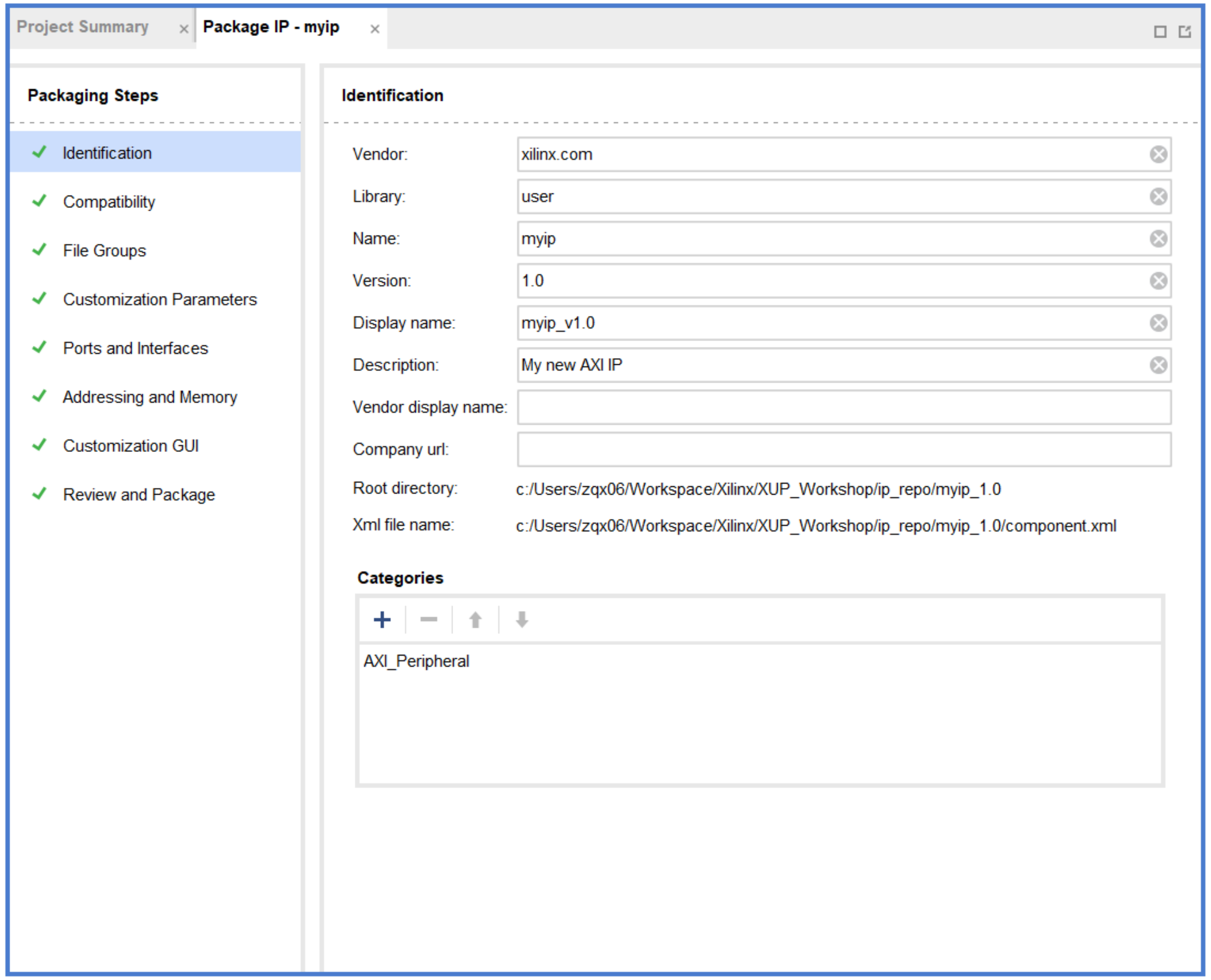
- Industry Standard (IEEE) XML format to describe IP using meta-data
  - Ports
  - Interfaces
  - Configurable Parameters
  - Files, documentation
- IP-XACT only describes high level information about IP, not low level description, so does not replace HDL or Software.
- Enables automatic connection, configuration and integration
- Enables integration of 3rd Party IP
  - (And Export of your own IP)

IP Packager

- Automatically analyse project/files to determine parameters
- Initial Summary
- Identifies
  - Files
    - Source HDL, Testbenches, Documentation,
  - Parameters
    - Configurable
  - Ports
  - Interfaces
  - Compatibility
- Creates GUI Layout for IPI

IP Packager

- Modify configuration
  - Properties
  - Compatibility
  - Files
  - Custom parameters
  - Ports
  - Interfaces
  - Address and Memory
  - IP and security
- Project can be updated – e.g., source files added
  - Changes will be reflected in packager
- Review and package



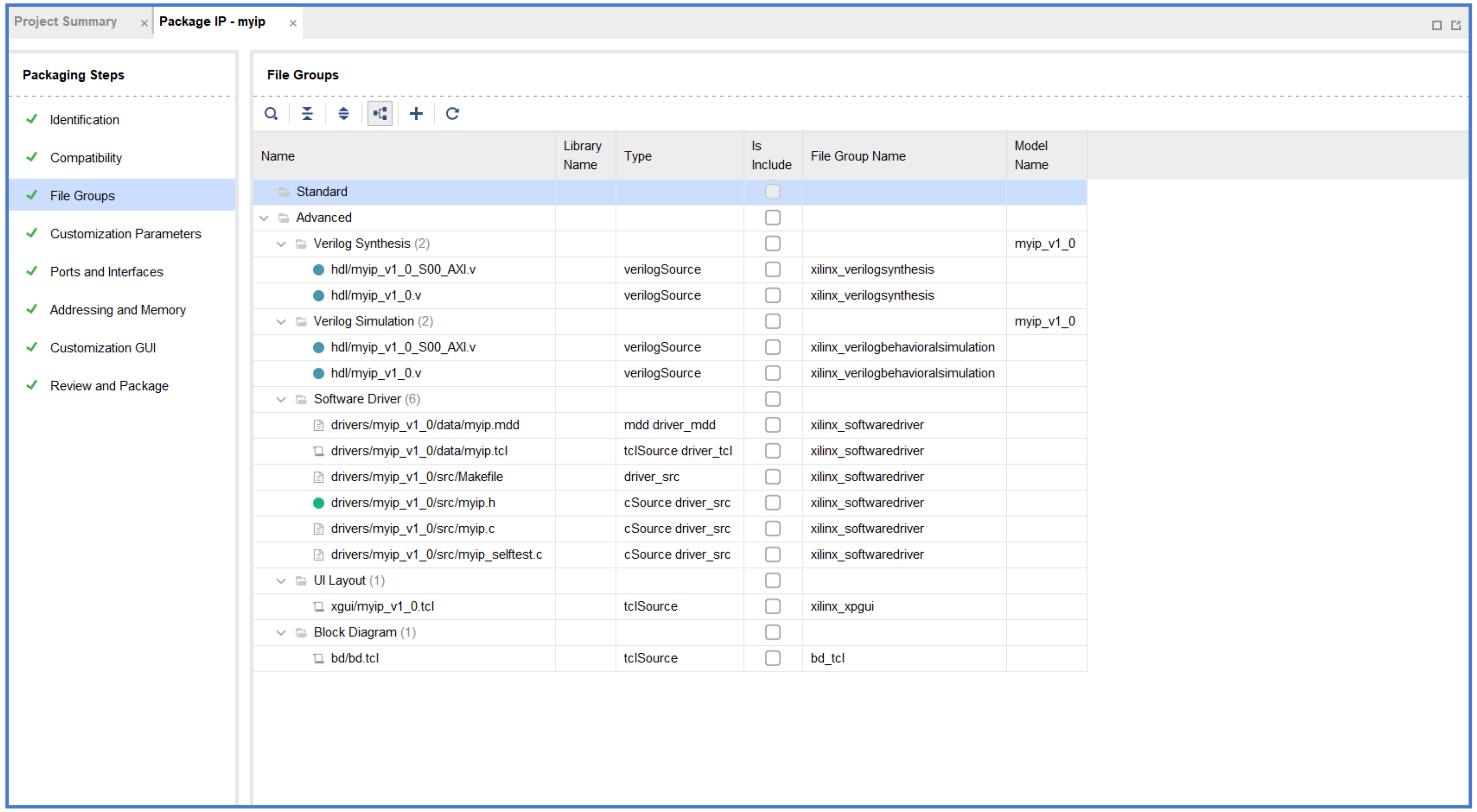
Identification

Vendor:	xilinx.com
Library:	user
Name:	myip
Version:	1.0
Display name:	myip_v1.0
Description:	My new AXI IP
Vendor display name:	
Company url:	
Root directory:	c:/Users/zqx06/Workspace/Xilinx/XUP_Workshop/ip_repo/myip_1.0
Xml file name:	c:/Users/zqx06/Workspace/Xilinx/XUP_Workshop/ip_repo/myip_1.0/component.xml

Categories

AXI_Peripheral

Customizing IP for Reuse in IP Packager



Project Summary | Package IP - myip

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- ✓ Addressing and Memory
- ✓ Customization GUI
- ✓ Review and Package

File Groups

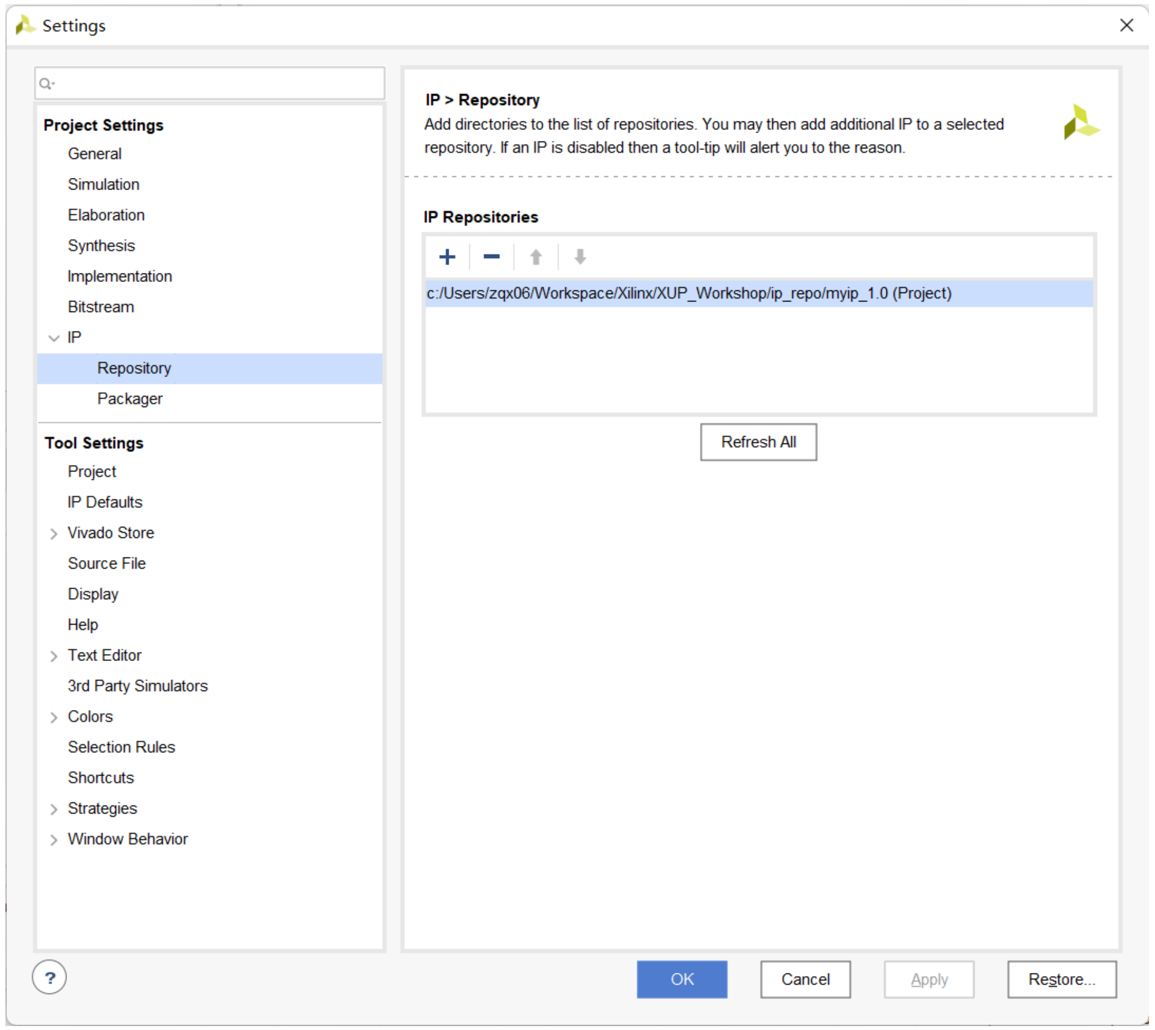
Name	Library Name	Type	Is Include	File Group Name	Model Name
Standard			☐		
Advanced			☐		
Verilog Synthesis (2)			☐		myip_v1_0
hdl/myip_v1_0_S00_AXI.v		verilogSource	☐	xilinx_verilogsynthesis	
hdl/myip_v1_0.v		verilogSource	☐	xilinx_verilogsynthesis	
Verilog Simulation (2)			☐		myip_v1_0
hdl/myip_v1_0_S00_AXI.v		verilogSource	☐	xilinx_verilogbehavioralsimulation	
hdl/myip_v1_0.v		verilogSource	☐	xilinx_verilogbehavioralsimulation	
Software Driver (6)			☐		
drivers/myip_v1_0/data/myip.mdd		mdd driver_mdd	☐	xilinx_softwaredriver	
drivers/myip_v1_0/data/myip.tcl		tclSource driver_tcl	☐	xilinx_softwaredriver	
drivers/myip_v1_0/src/Makefile		driver_src	☐	xilinx_softwaredriver	
drivers/myip_v1_0/src/myip.h		cSource driver_src	☐	xilinx_softwaredriver	
drivers/myip_v1_0/src/myip.c		cSource driver_src	☐	xilinx_softwaredriver	
drivers/myip_v1_0/src/myip_selftest.c		cSource driver_src	☐	xilinx_softwaredriver	
UI Layout (1)			☐		
xgui/myip_v1_0.tcl		tclSource	☐	xilinx_xpgui	
Block Diagram (1)			☐		
bd/bd.tcl		tclSource	☐	bd_tcl	

Select Options

Add, Edit or change defaults

IP repository

- Extend the Vivado IP Catalog by adding additional IP Repositories
- Third party IP, your custom IP, and Xilinx IP are displayed in an identical manner
- Packager creates .xml file for the IP
- Specify the directory of the user/third party IP repository
  - Can be done automatically from packager
  - Access from project settings, or IP Catalog
  - Displays the IP that has been found
- Displays IP in the repository for use in IPI



IP > Repository
Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

IP Repositories

c:/Users/zqx06/Workspace/Xilinx/XUP_Workshop/ip_repo/myip_1.0 (Project)

Summary

Summary

- AXI4 interface defines five channels
  - All channels use basic VALID/READY handshake to complete a transfer
- AXI Interconnect extends AXI interface by allowing 1-to-N, N-to-1, N-to-M, and M-to-N connections
- Custom IP can be imported using IP Packager
- Include in the IP Repository for reuse across projects
- Create and Package wizard supports AXI Lite, Full, and Stream compatible IP creation and packaging
  - Handles interface side protocol
  - Provides template to add HDL functionality
  - Packages into the IP Catalog

Thank You

Disclaimer and Attribution

AMD
XILINX