

Creating and Adding Custom IP



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

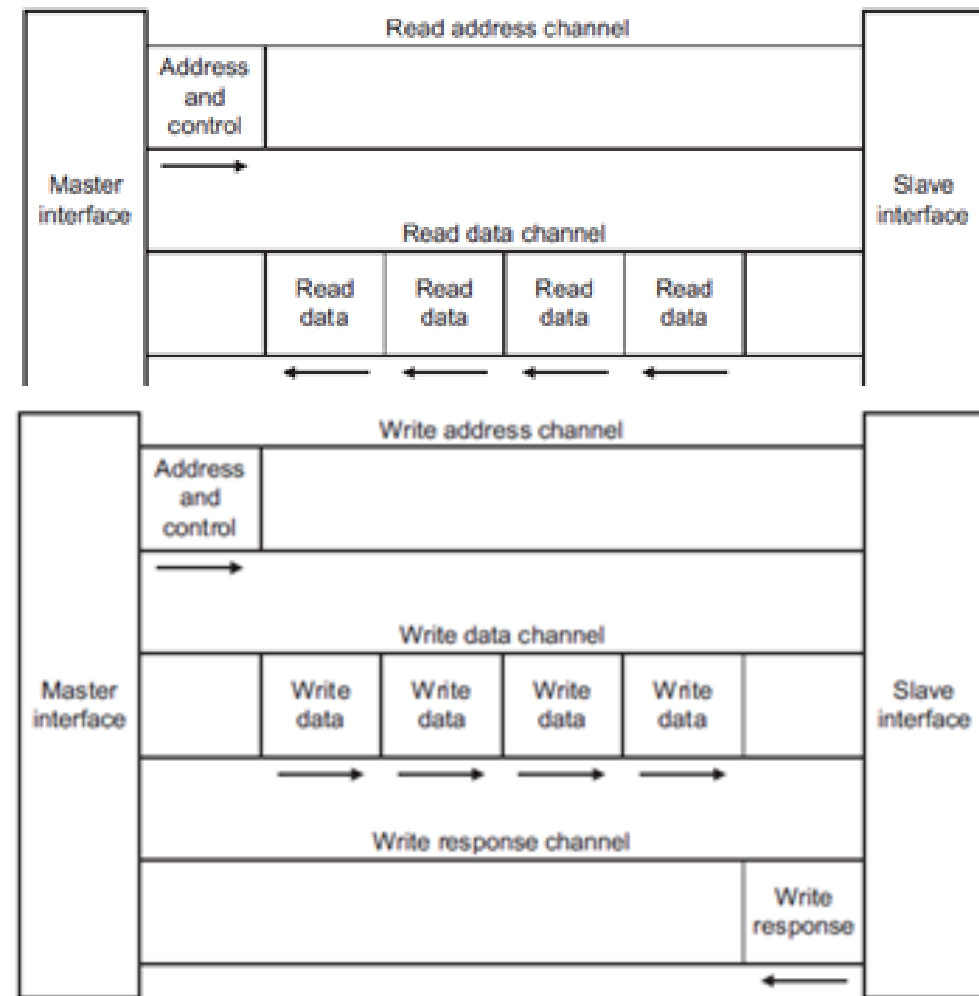
Outline

- > ***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"

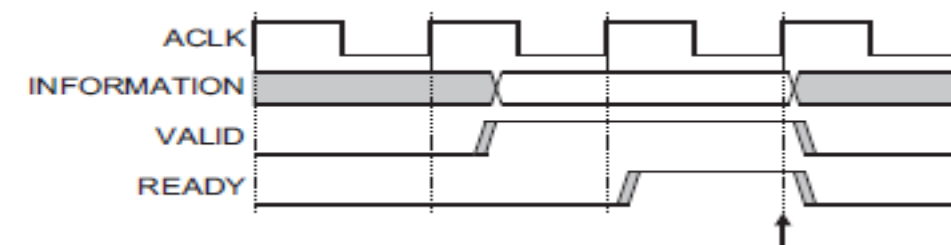


All AXI Channels Use A Basic “VALID/READY” Handshake

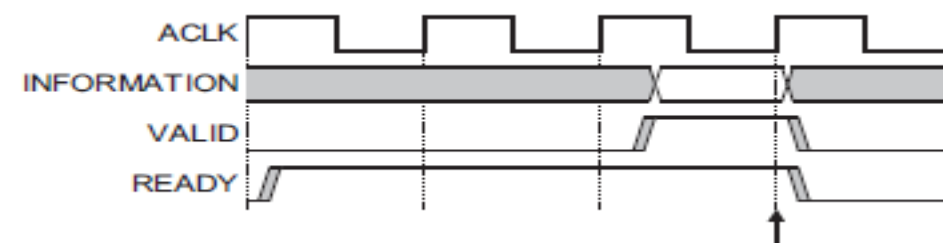
- > ***SOURCE*** asserts and holds **VALID** when **DATA** is available
- > ***DESTINATION*** asserts **READY** if able to accept **DATA**
- > **DATA** transferred when **VALID** and **READY** = 1
- > ***SOURCE*** sends next **DATA** (if an actual data channel) or deasserts **VALID**
- > ***DESTINATION*** deasserts **READY** if no longer able to accept **DATA**

AXI Interface: Handshaking

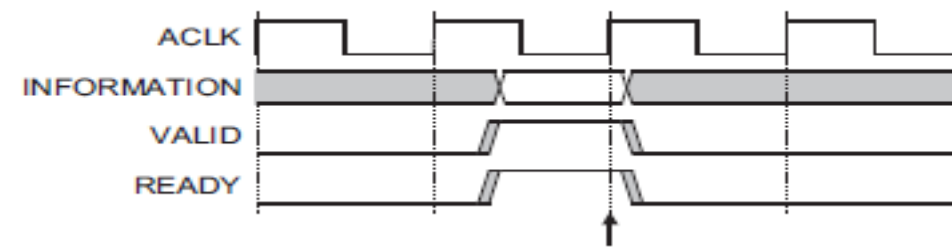
- > **AXI uses a valid/ready handshake acknowledge**
- > **Each channel has its own valid/ready**
 - >> Address (read/write)
 - >> Data (read/write)
 - >> Response (write only)
- > **Flexible signaling functionality**
 - >> Inserting wait states
 - >> Always ready
 - >> Same cycle acknowledge



Inserting Wait States



Always Ready



Same Cycle Acknowledge

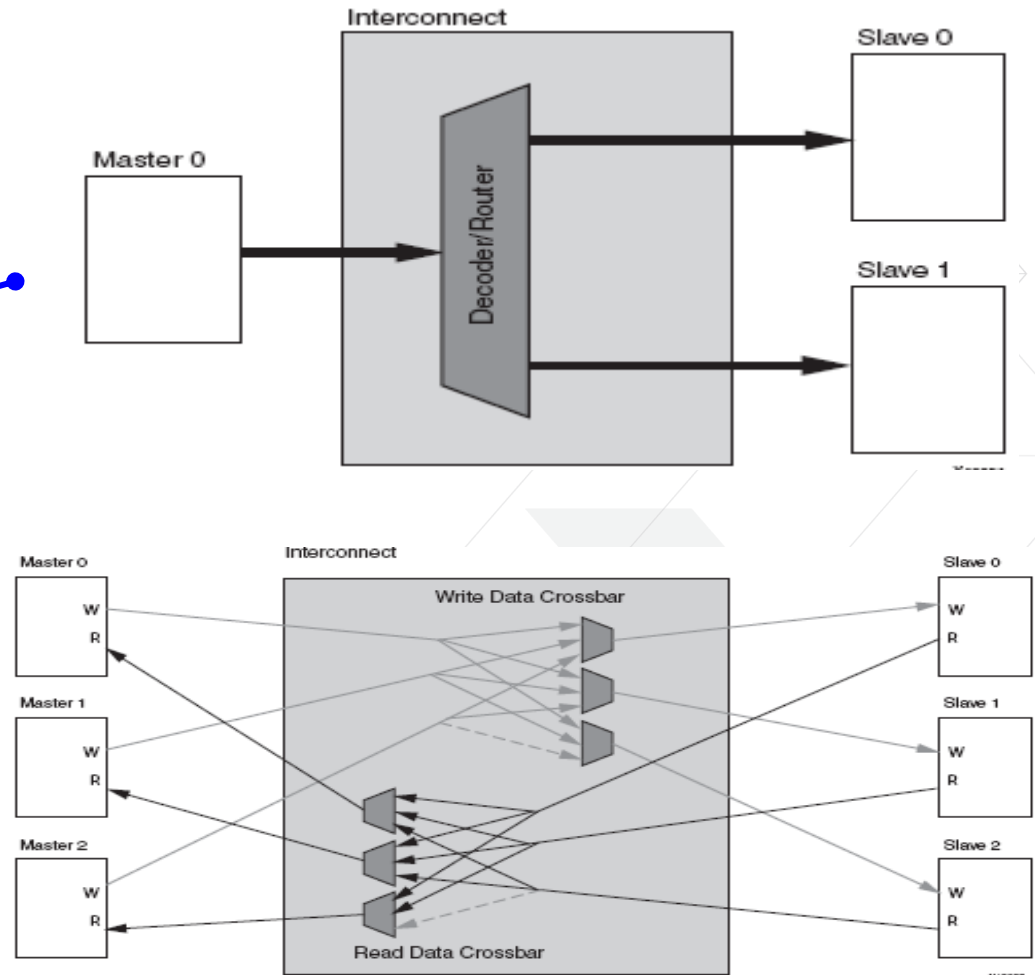
AXI Interconnect

> 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 Signals (AXI4, AXI4-Lite)

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

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

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

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

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

AXI4 signals

- > ID – Identification signal
- > LEN – Burst length; 256 INCR, 16 Wrap
- > Size – Transfer width 8-256 bytes
- > Burst – INCR, WRAP, FIXED
- > Lock – Exclusive, atomic access
- > CACHE –Transaction caching
- > PROT – Access Protection
- > QOS – QOS signal
- > Region – allows multiple logical interfaces on slave
- > User – User signal
- > STRB – strobe
- > LAST – Indicate last transaction in burst

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

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

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

See AMBA® AXI™ and ACE™ Protocol Specification

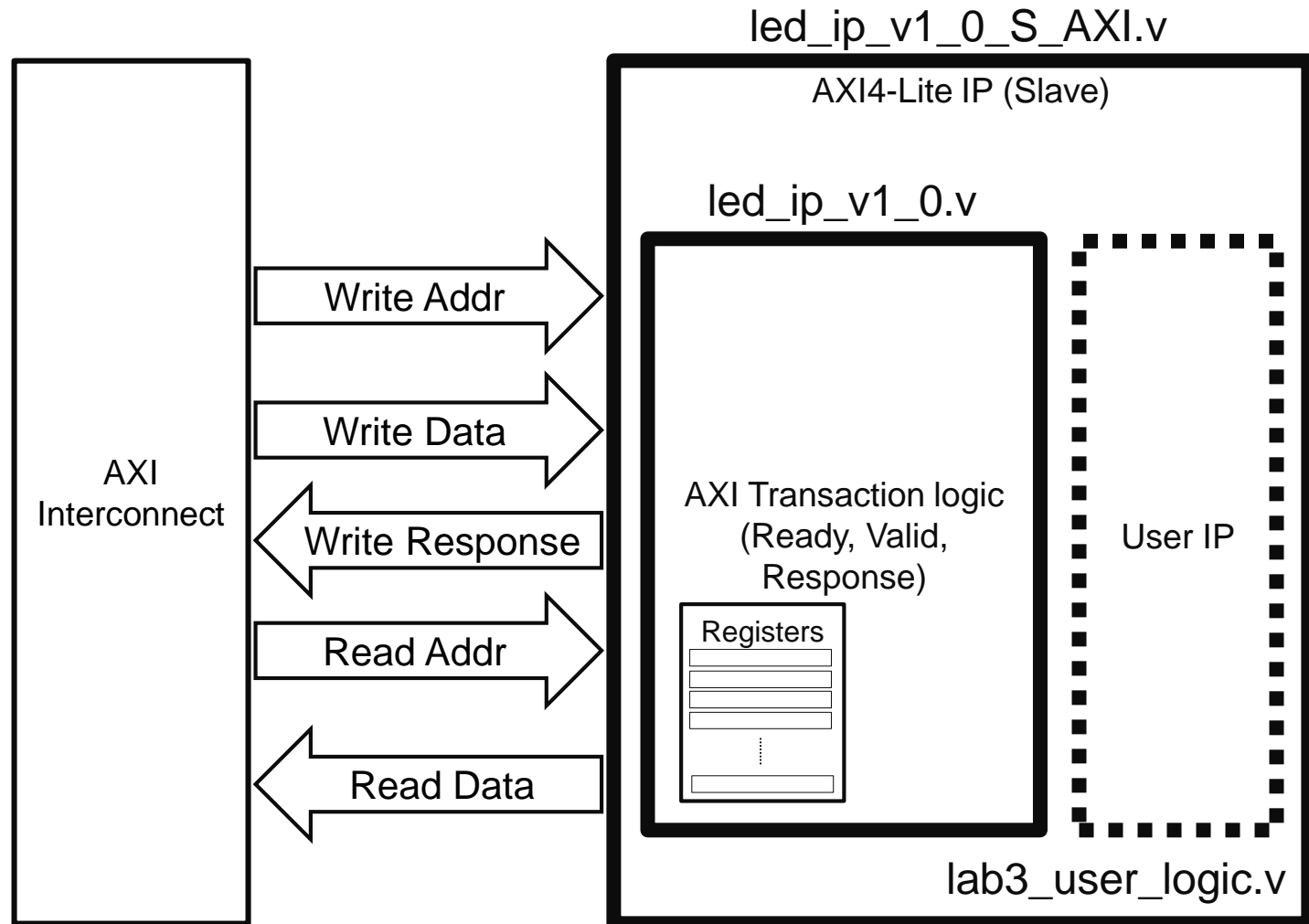
Outline

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



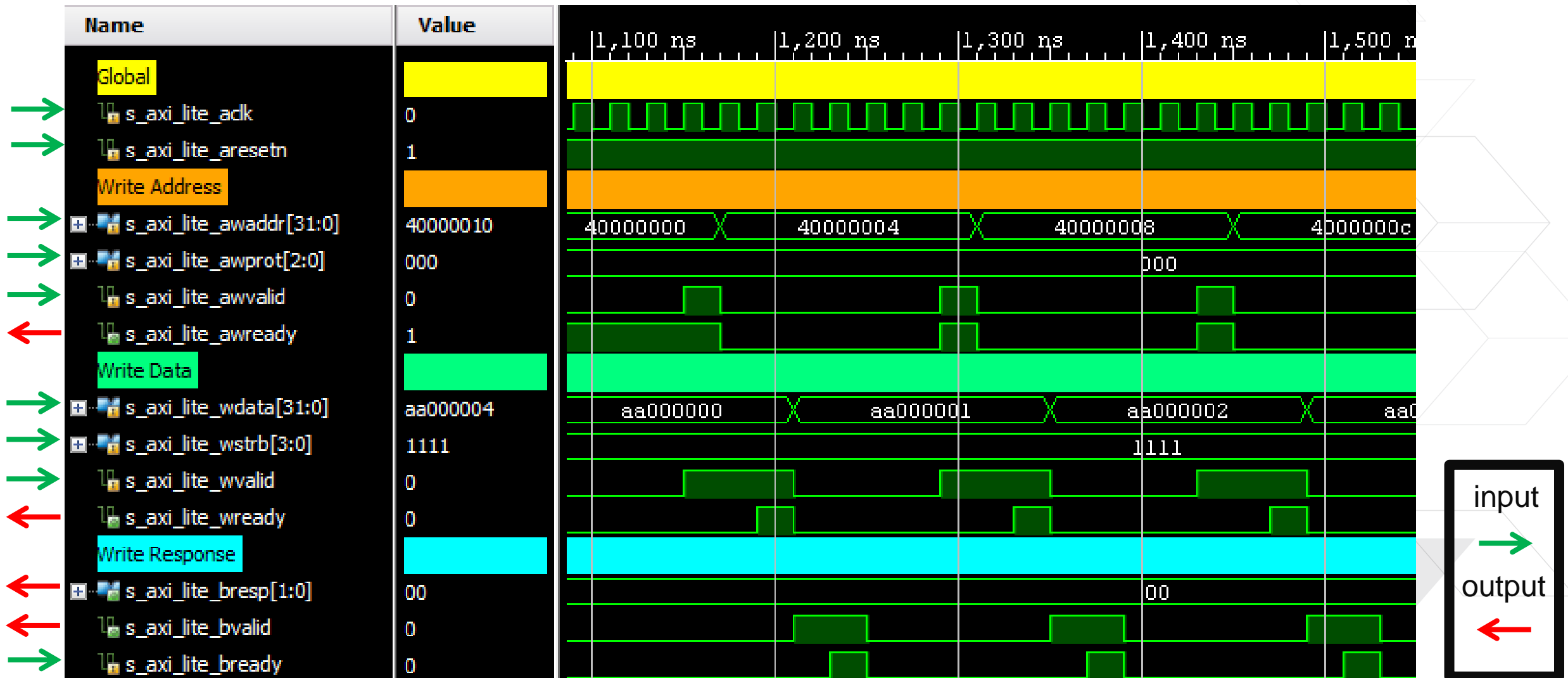
AXI4-Lite Slave

- > **AXI4-Lite is a subset of AXI (Full)**
 - >> Same channels, but less signals than full interface
- > **5 AXI channels**
 - >> Inputs: Write Address, Data; Read Address
 - >> Outputs: Write Response; Read Data
- > **Slave responds to read and write transactions from upstream Master**
- > **1 data transfer per transaction**
 - >> No burst support
- > **Intended for lower performance peripherals**



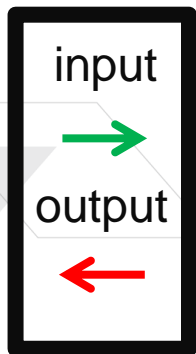
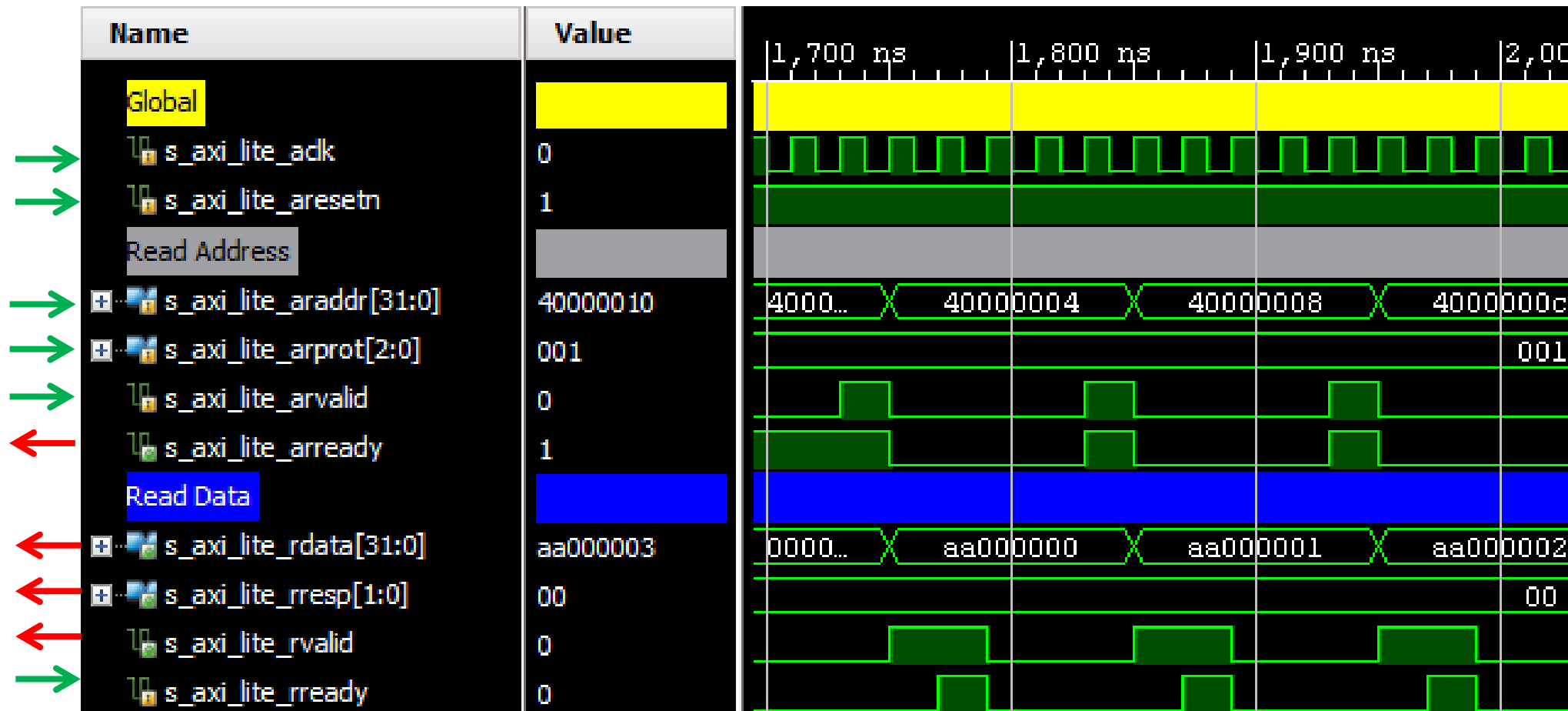
AXI Lite Slave

Single Data Phase Write



AXI Lite Slave

Single Data Phase Read



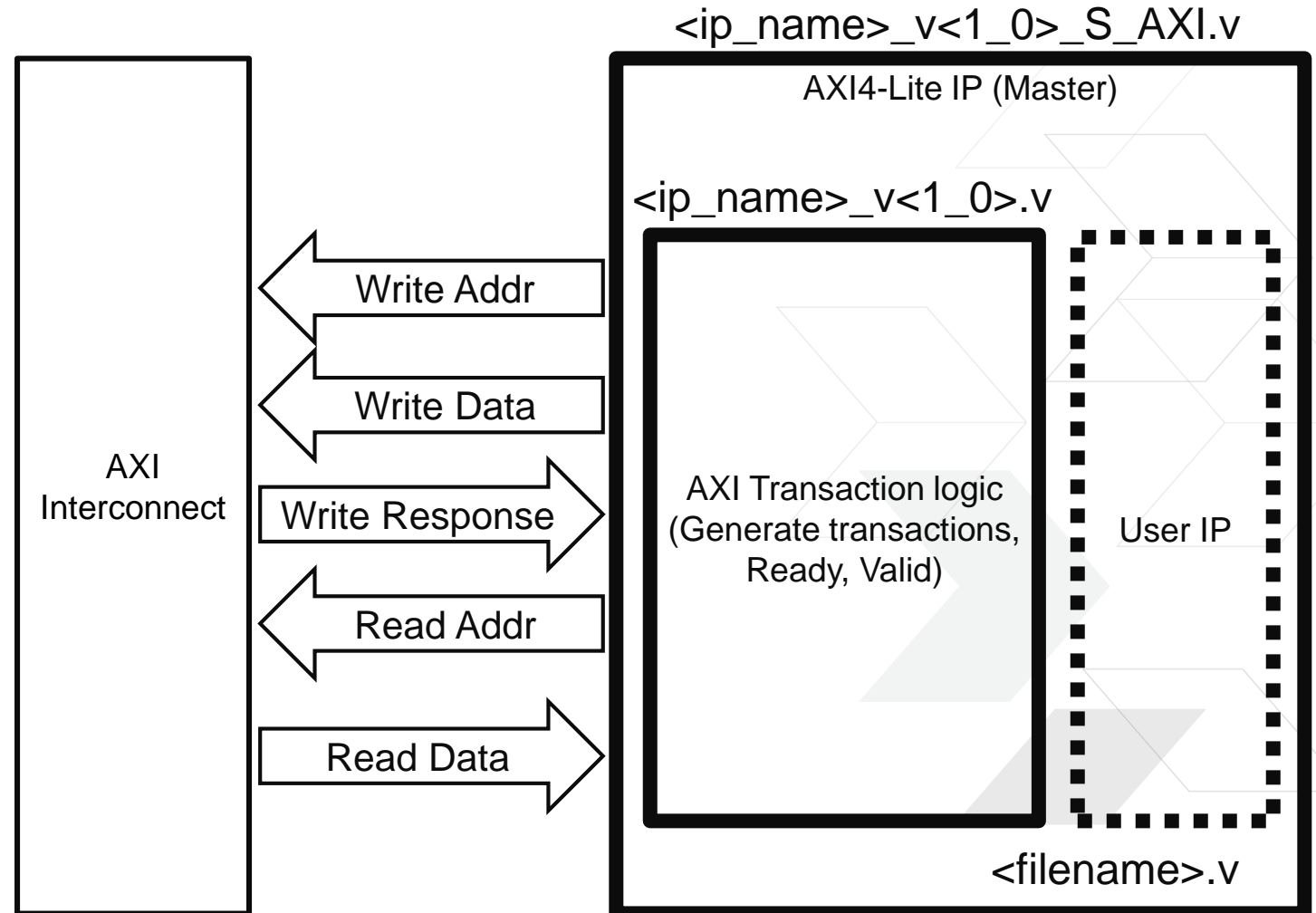
Outline

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



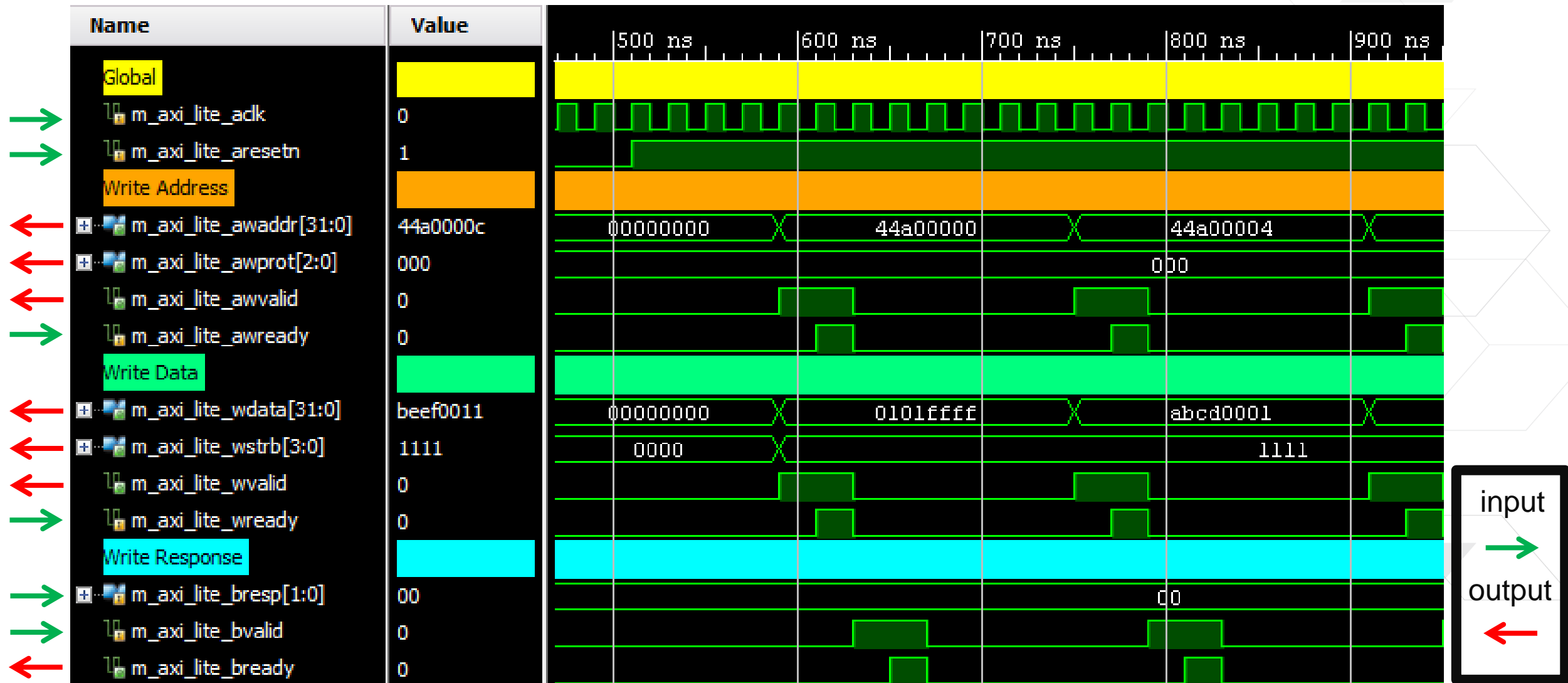
AXI4-Lite Master

- > **5 AXI channels**
 - >> Outputs: Write Address, Data; Read Address
 - >> Inputs: Write Response; Read Data
- > **Opposite directions to Slave**
- > **Master can initiate read and write transactions**
- > **1 data transfer per transaction**
 - >> No burst support
- > **Addressable register bank in slave**



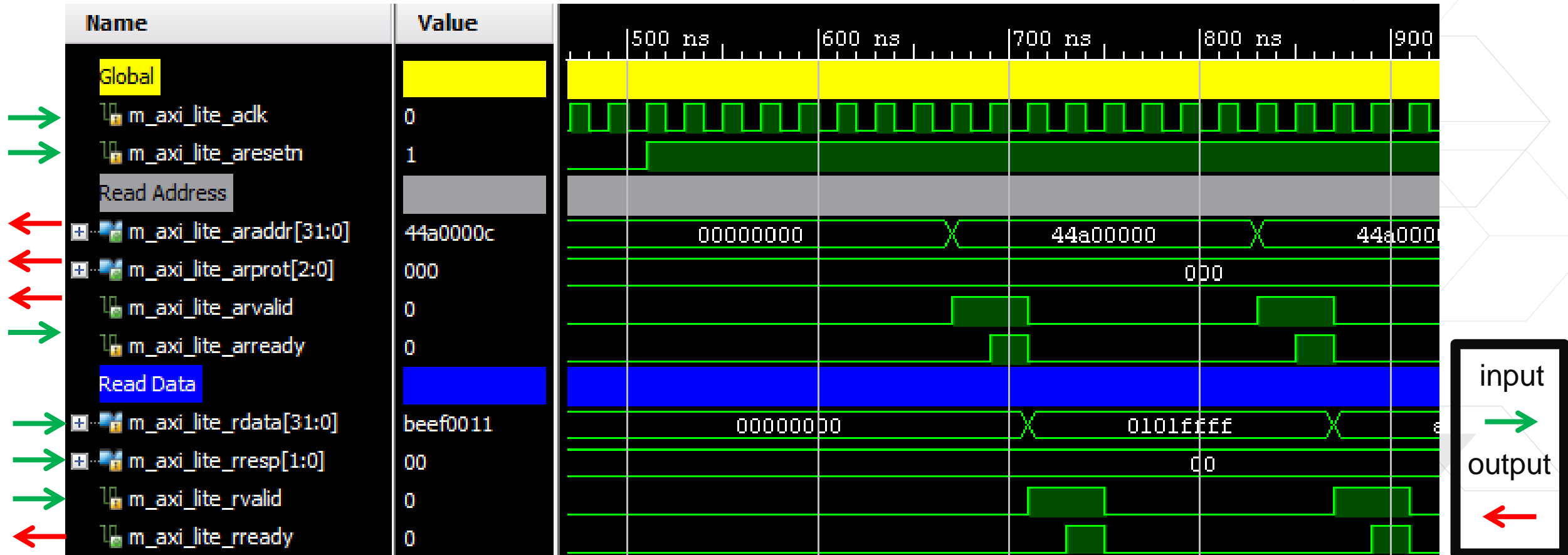
AXI4-Lite Master

Single Data Phase Write



AXI4-Lite Master

Single Data Phase Read



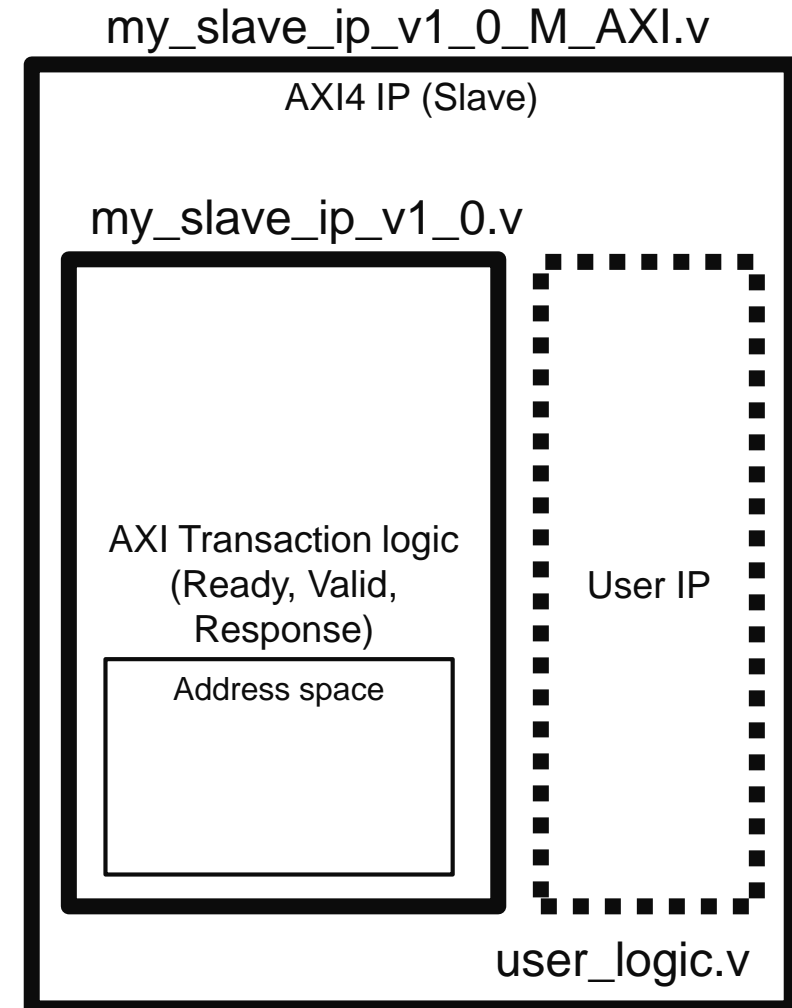
Outline

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



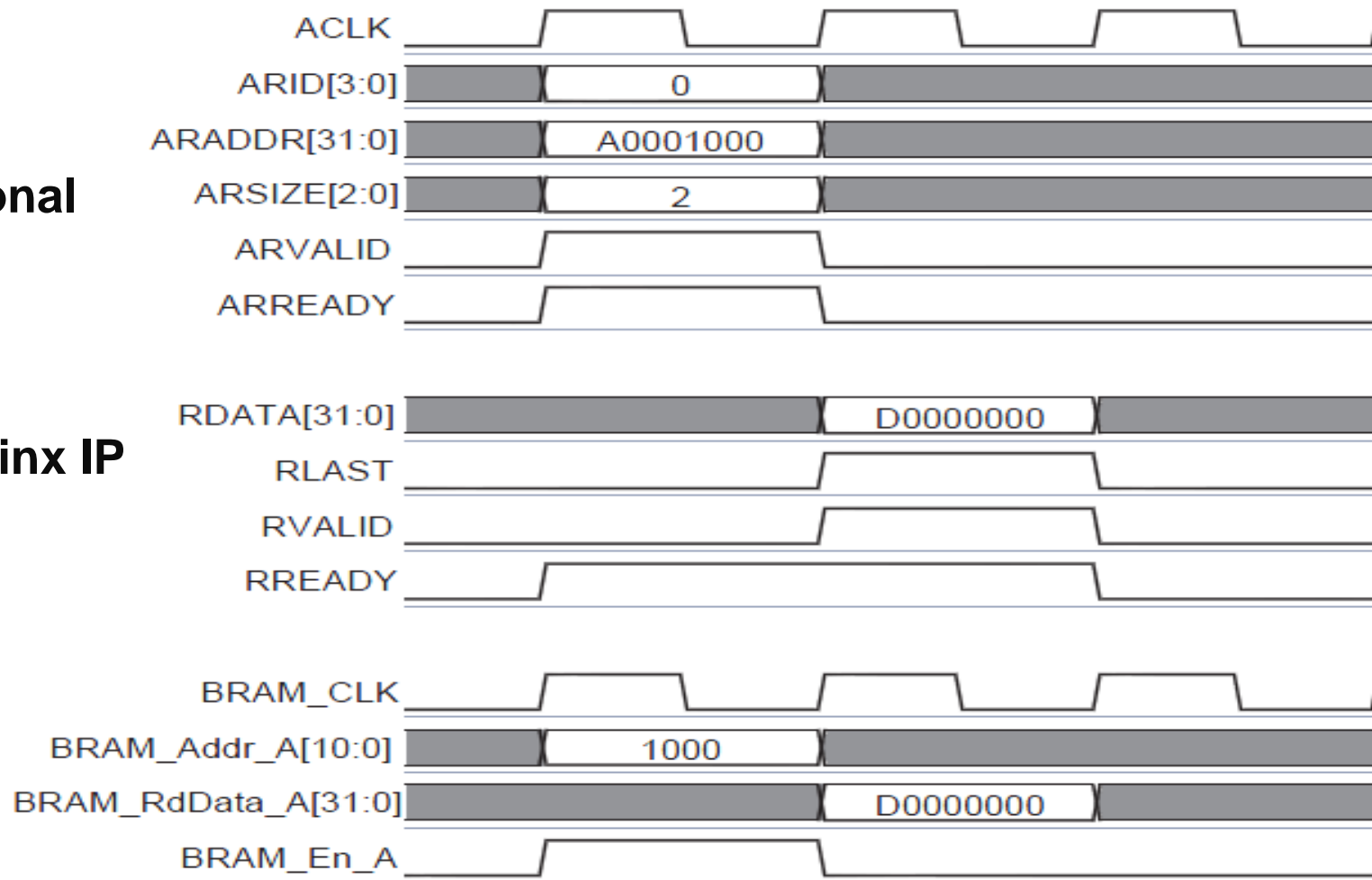
AXI4

- > **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
- > **Memory space**



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

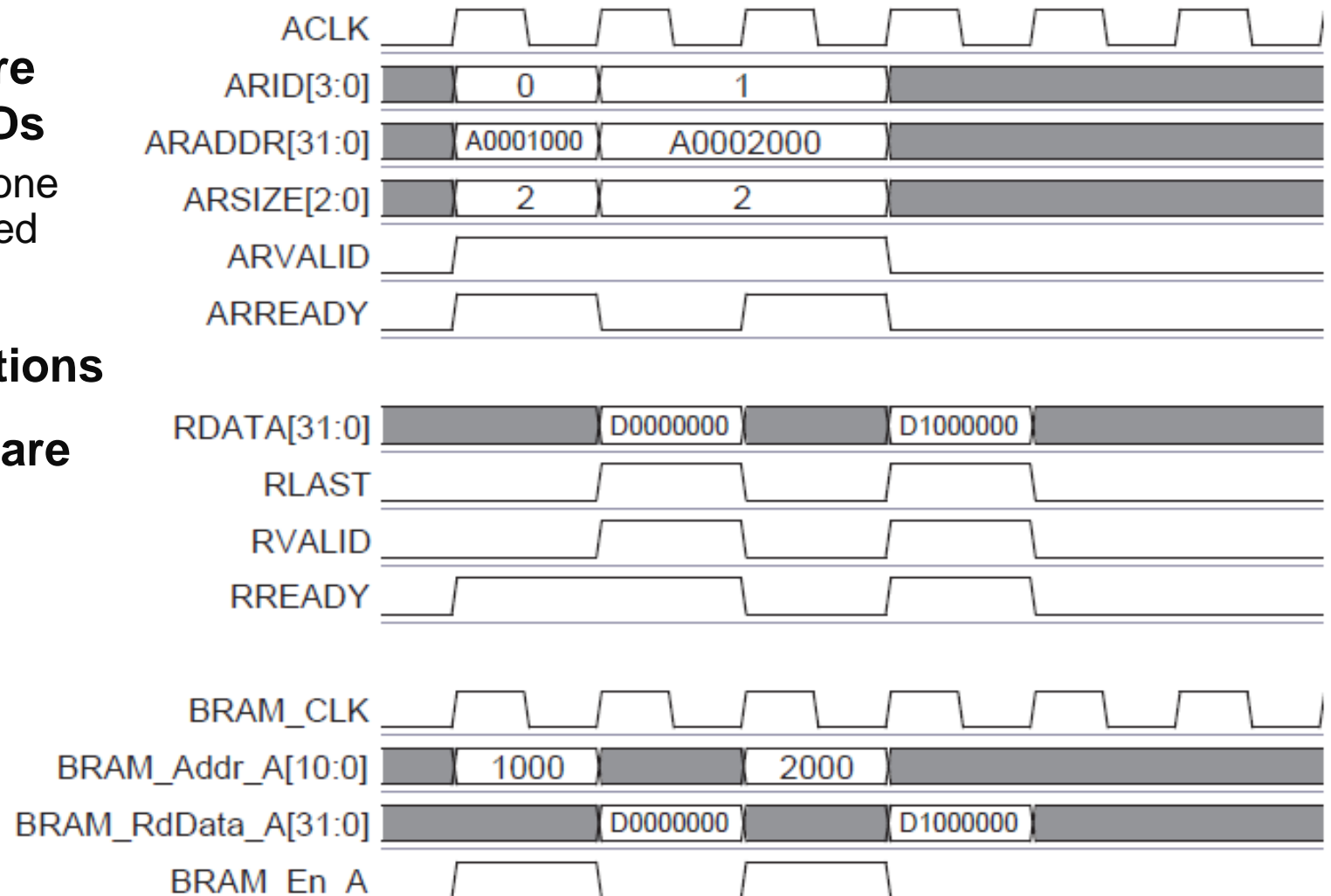


Example of read from AXI4 BRAM

AXI4 Read Transaction

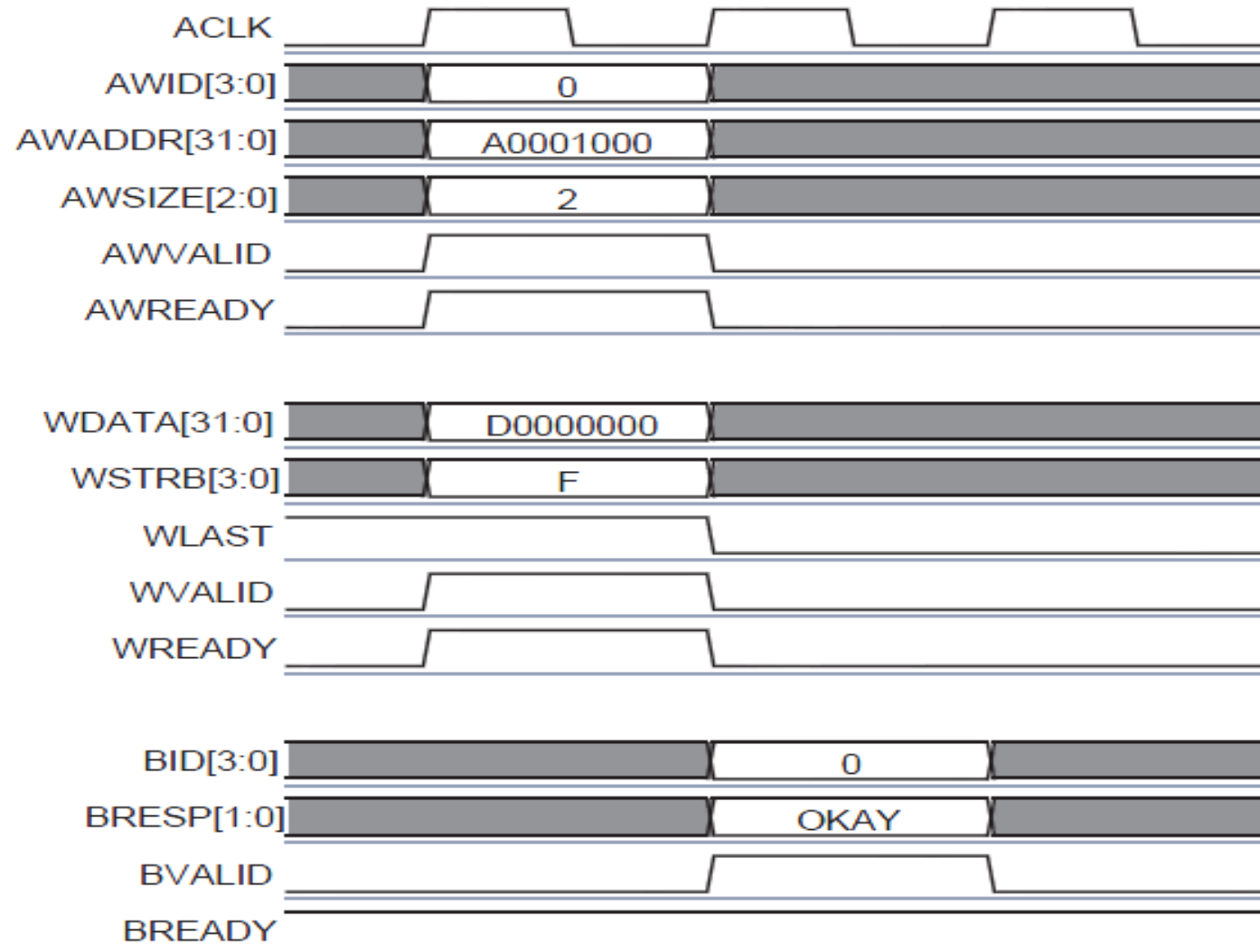
Multiple reads, not burst transaction

- > **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

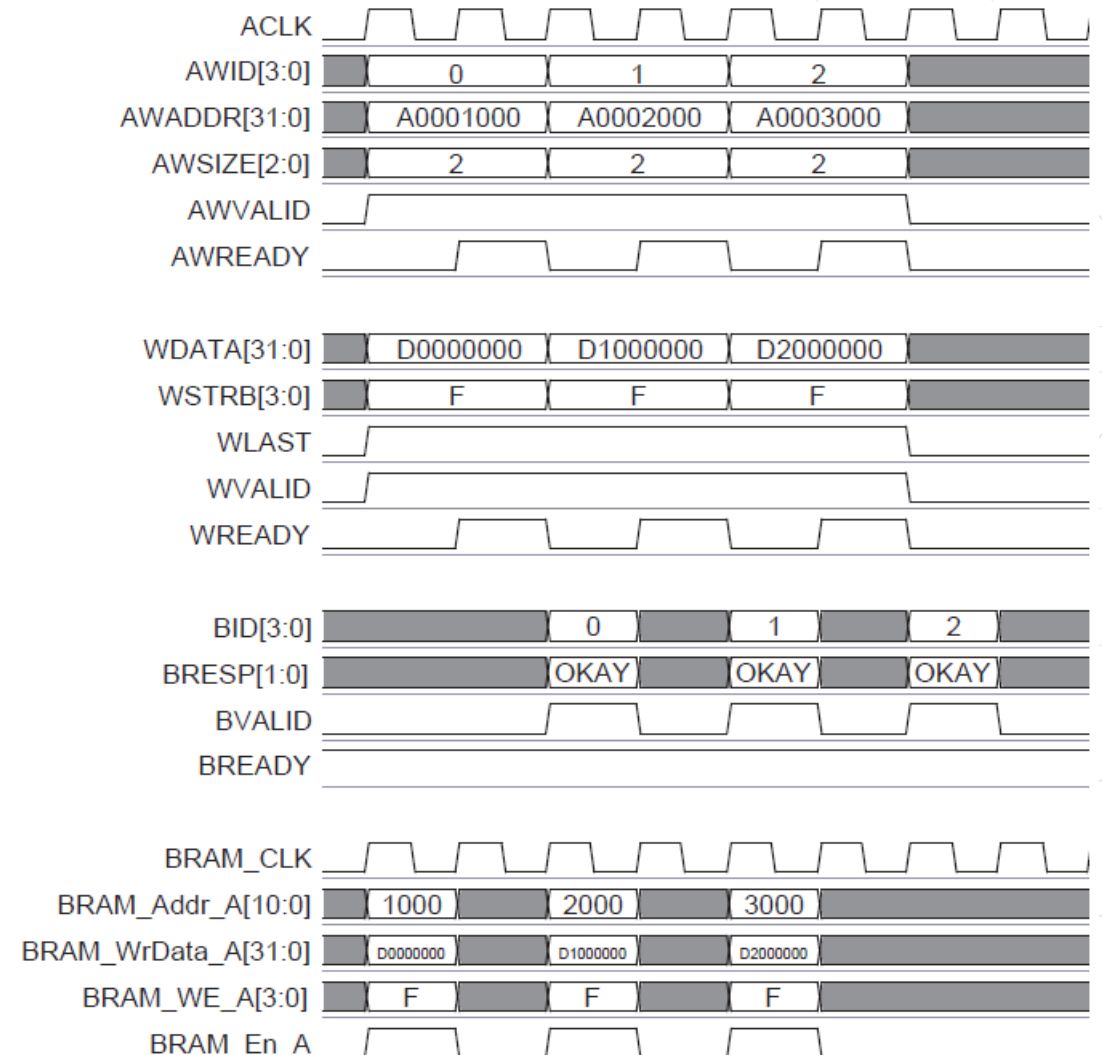


Example of AXI4 write transaction

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

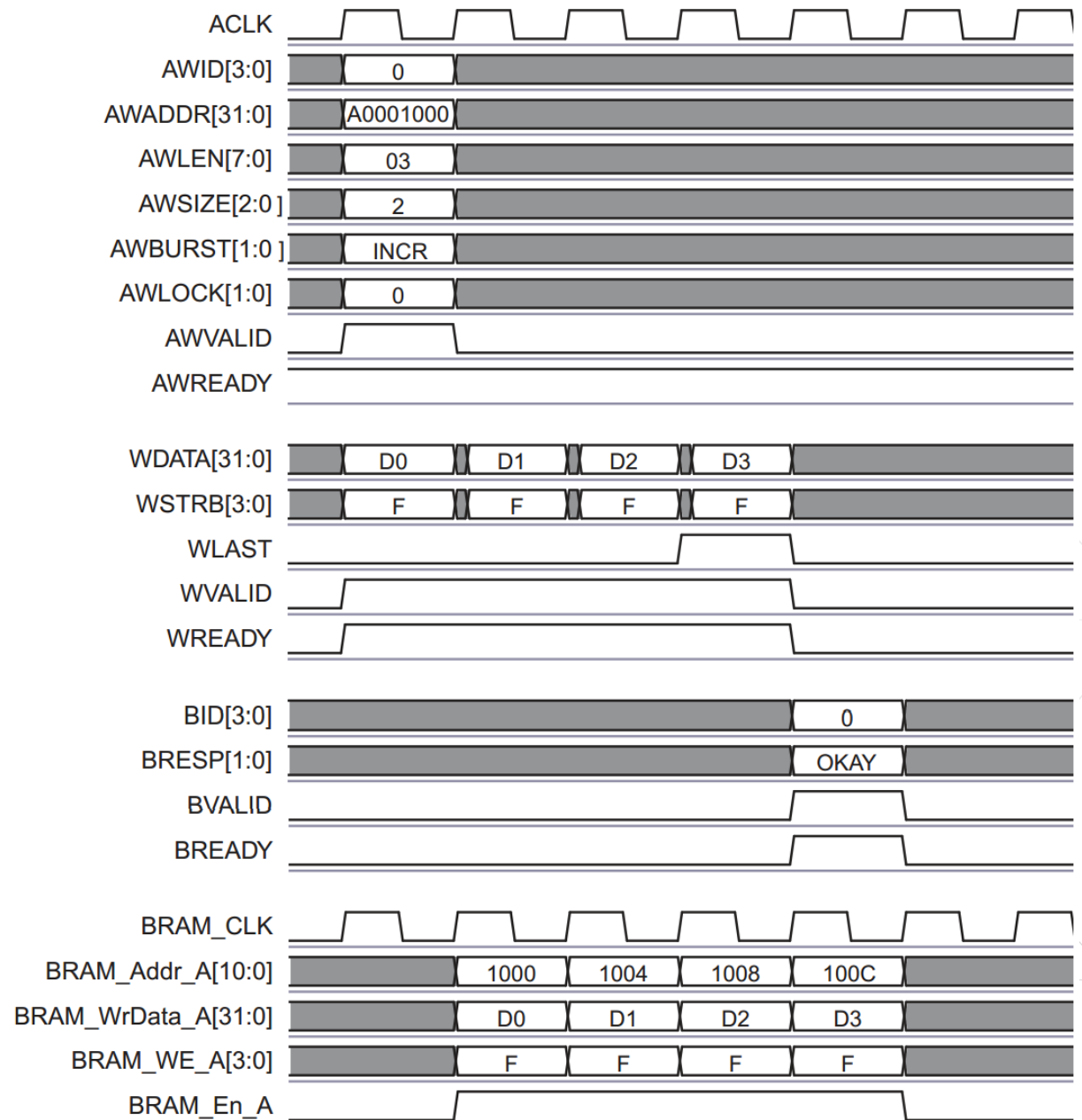


Example of AXI4 multiple write transactions

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**



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>



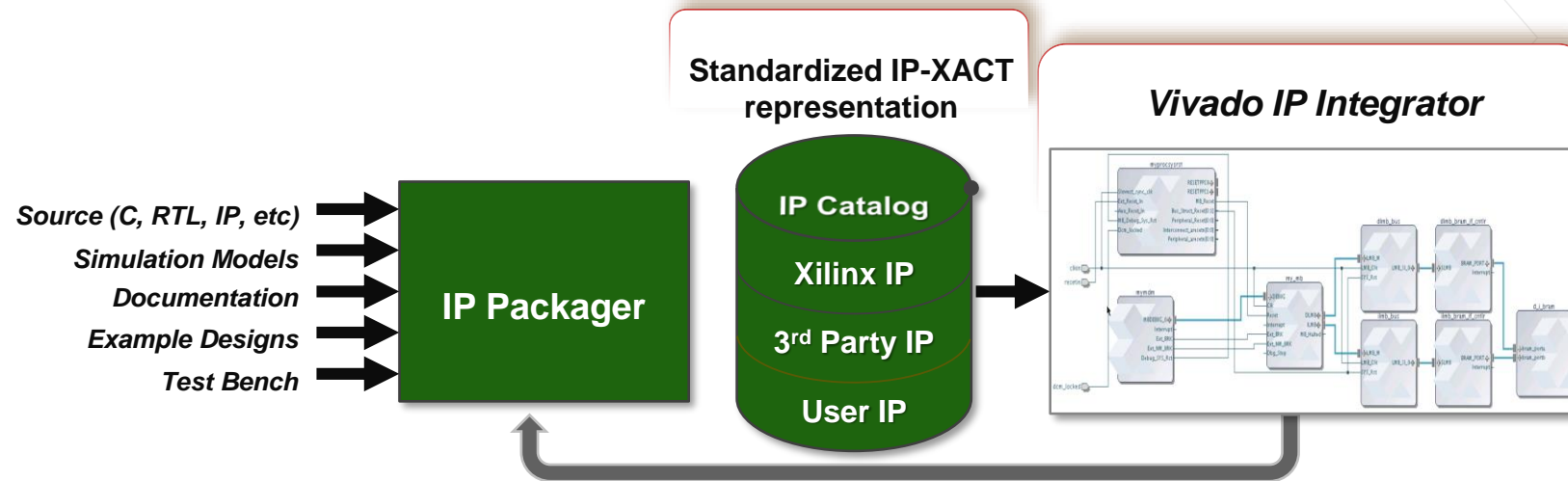
Outline

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



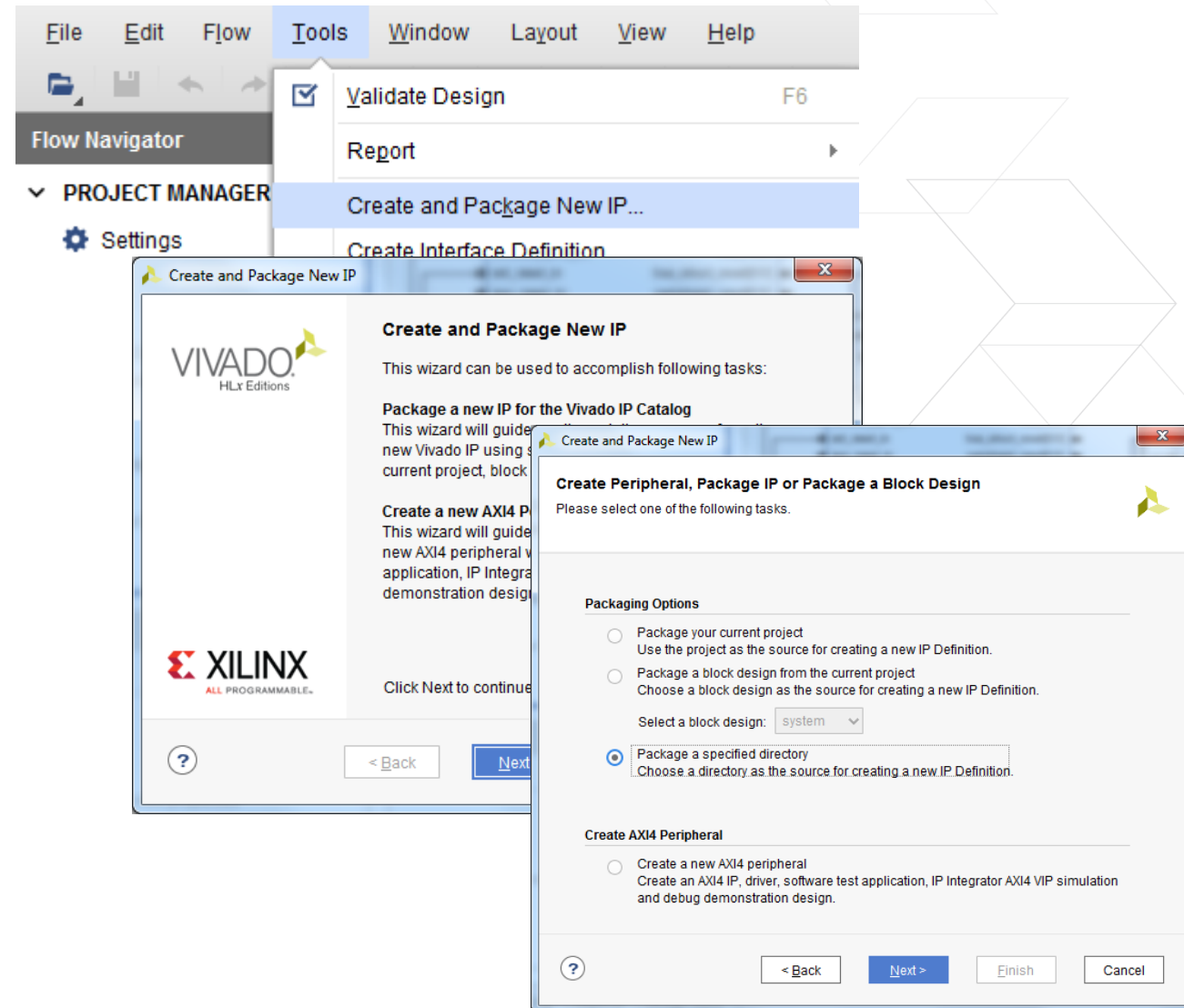
Reusing Your IP

- > 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



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



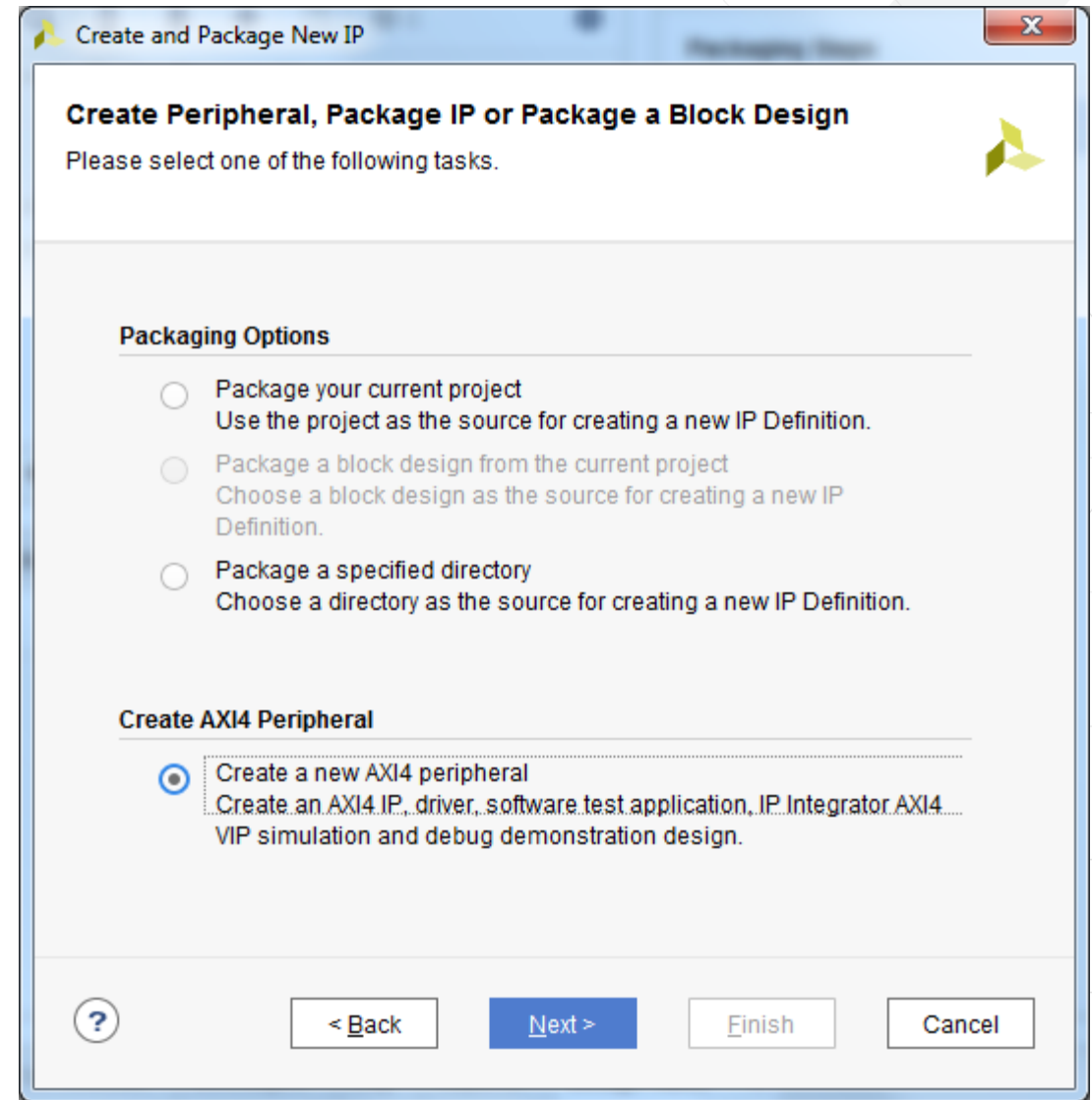
Outline

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



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 Custom IP

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



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_index++)
        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++)
        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_index++)
        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_index++)
        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
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
      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
      end
  end
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 merge
 - >> 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
    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 functions
- >> Simple SelfTest() code

> hdl

- >> Verilog/VHDL (should be inside project)

> 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");

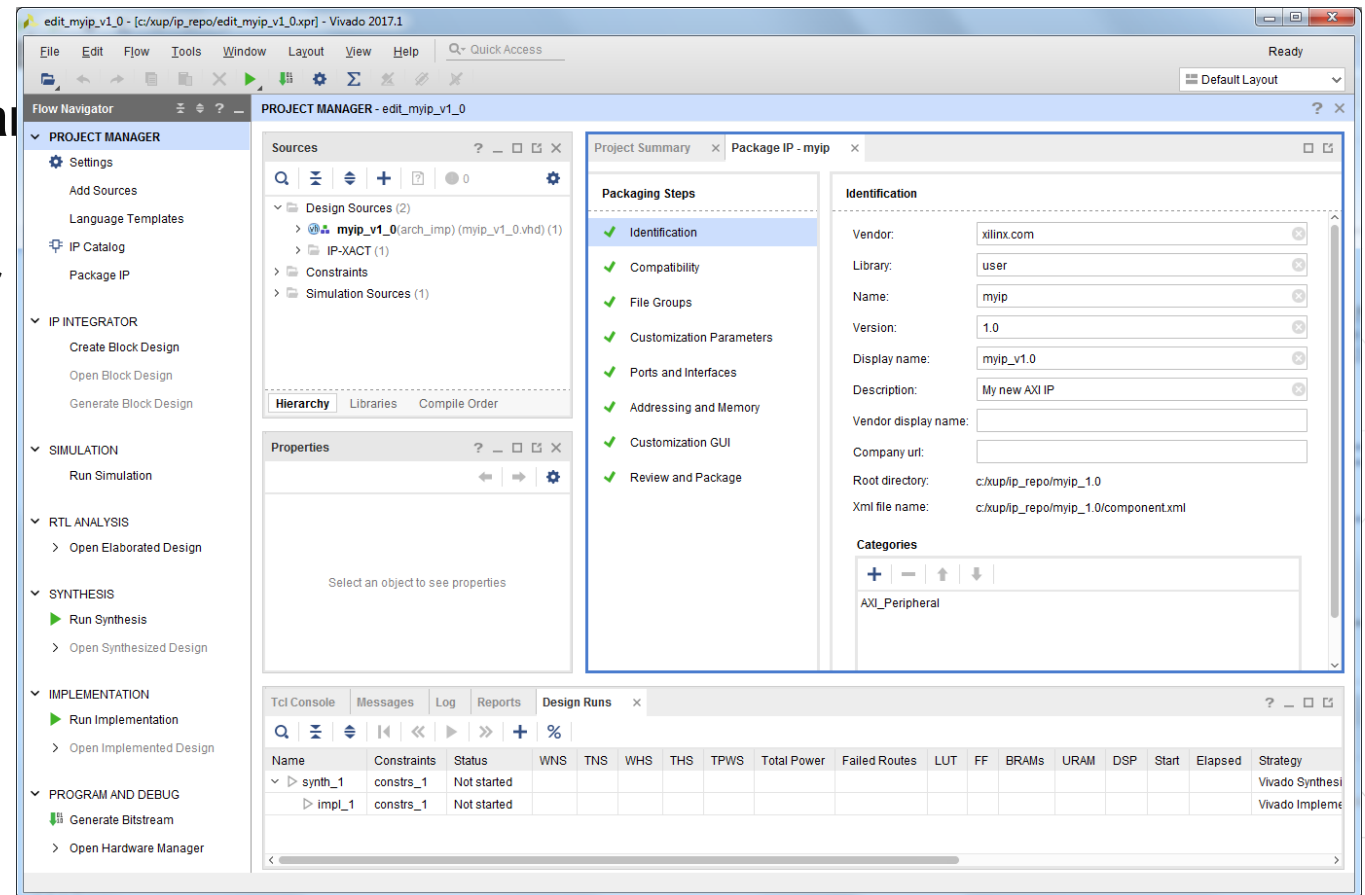
    /*
     * Write to user logic slave module register(s) and read back
     */
    xil_printf("User logic slave module test...\n\r");

    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_index+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
- > IP can now be edited
 - >> Modify existing template files, add user
- > IP Packager will be open
 - >> Last step is to Package IP



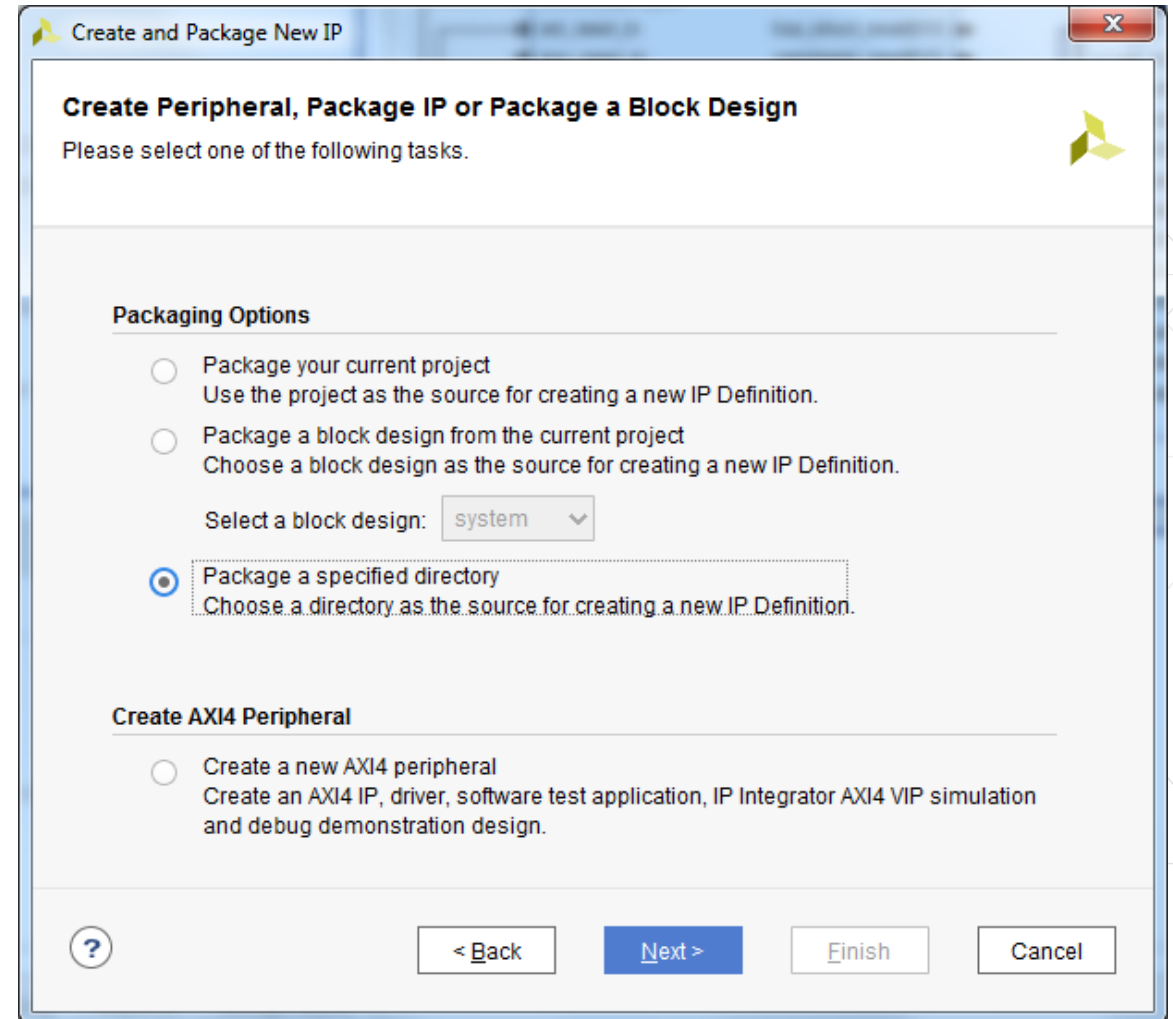
Outline

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



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



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**

Package IP - myip

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- ✓ Addressing and Memory
- ✓ Customization GUI
- ✓ 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:/xup/ip_repo/myip_1.0

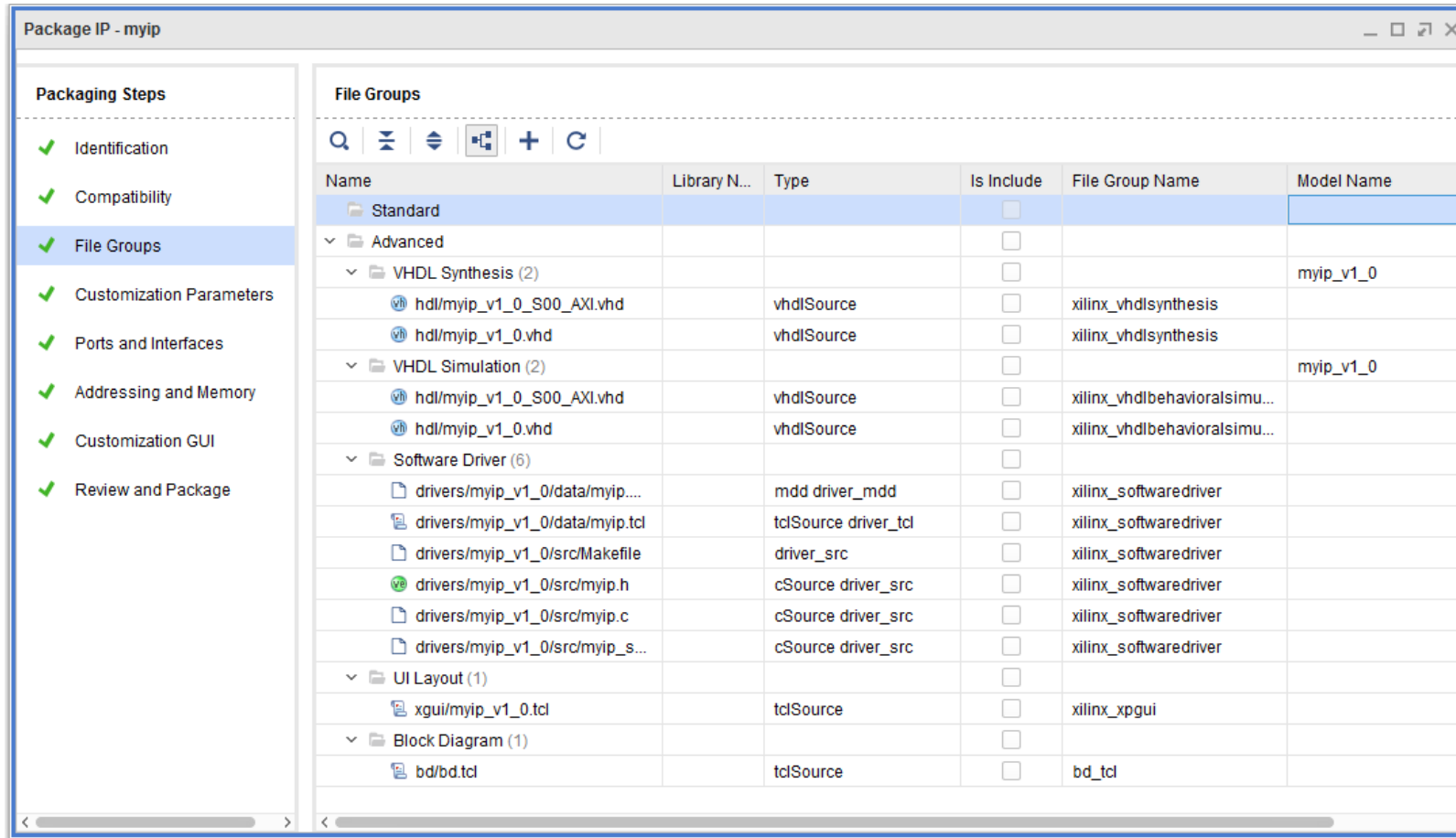
Xml file name: c:/xup/ip_repo/myip_1.0/component.xml

Categories

+ - ↑ ↓

AXI_Peripheral

Customizing IP for Reuse in IP Packager

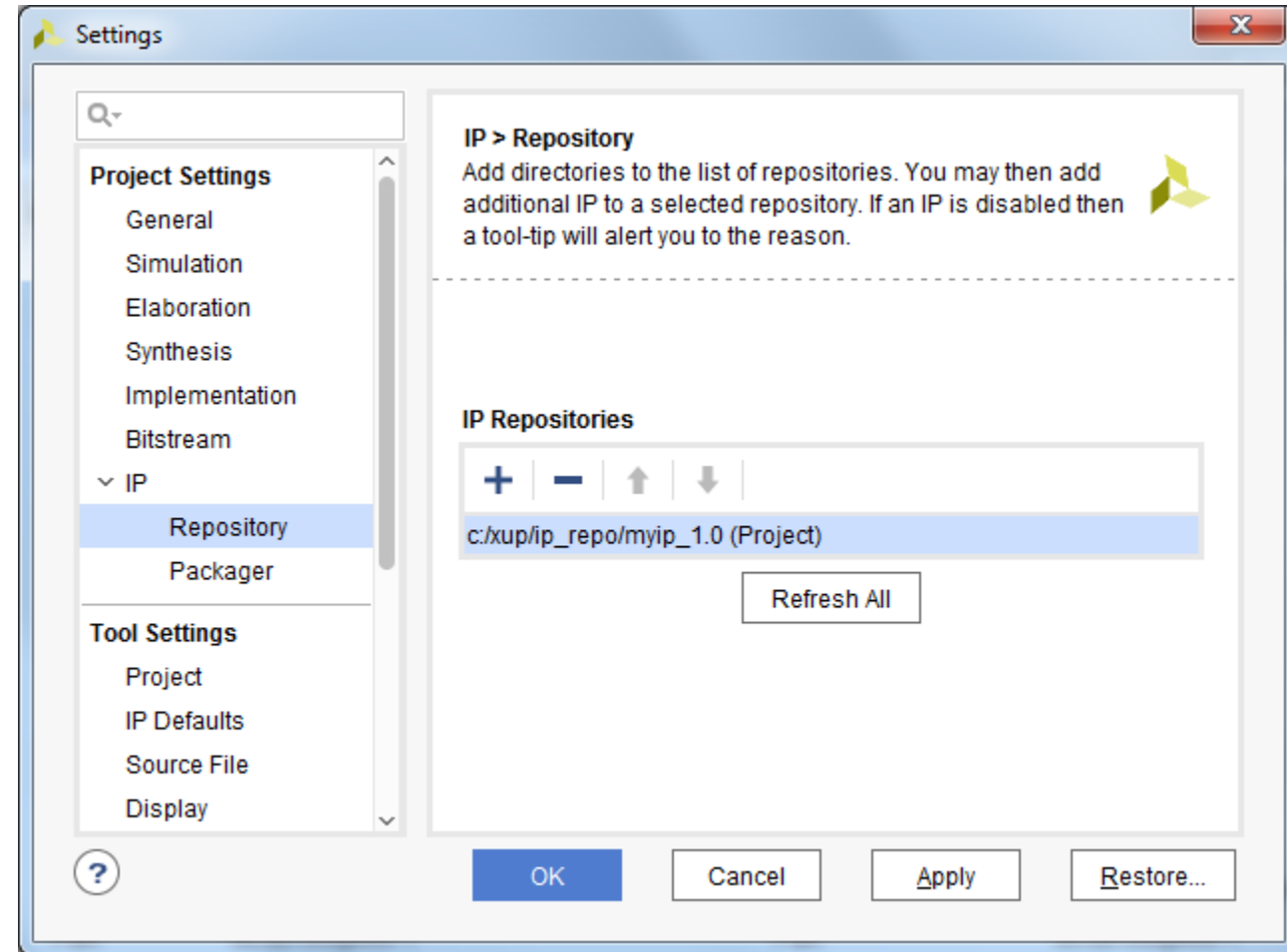


**Select
Options**

**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



Outline

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

Questions?

