Creating and Adding Custom IP

Zynq
Vivado 2013.3 Version

Objectives

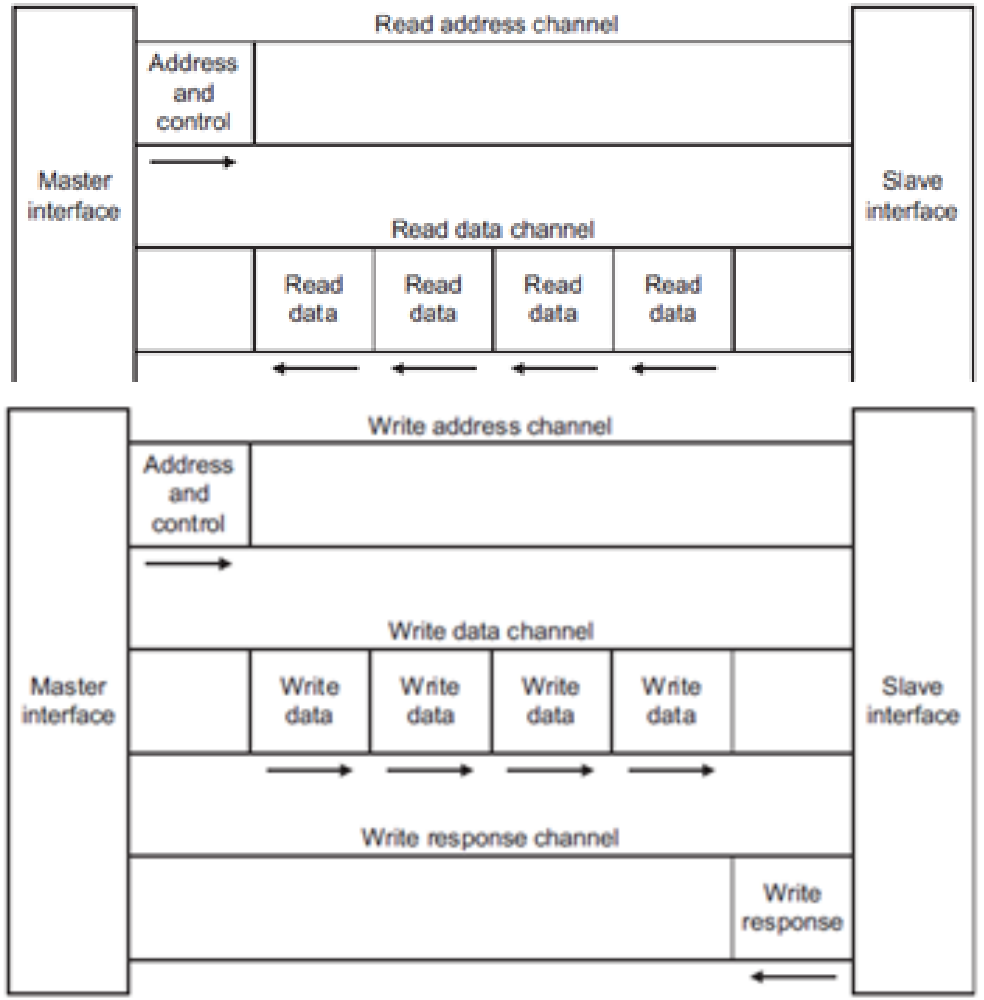
- **After completing this module, you will be able to:**
  - Describe the AXI4 transactions
  - Summarize the AXI4 valid/ready acknowledgment model
  - Discuss the AXI4 transactional modes of overlap and simultaneous operations
  - Describe the operation of the AXI4 streaming protocol
  - List the steps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

Outline

- ***AXI4 Transactions***
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - AXI4 Slave
  - AXI4 Master
- **IP Packager**
- **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

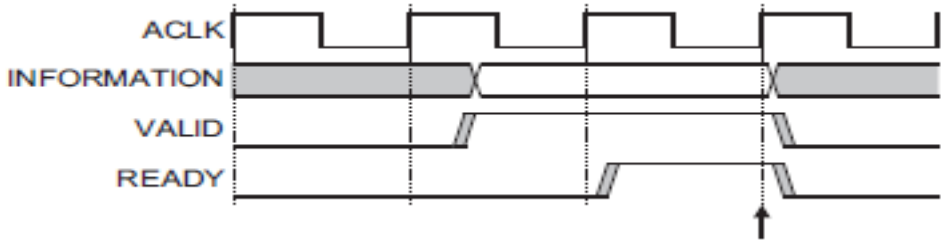
- *SOURCE* asserts and holds VALID when DATA is available
- *DESTINATION* asserts READY if able to accept DATA

- DATA transferred when VALID and READY = 1

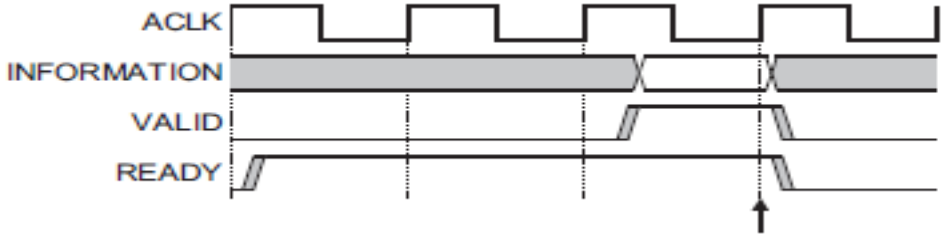
- *SOURCE* sends next DATA (if an actual data channel) or deasserts VALID
- *DESTINATION* deasserts READY if no longer able to accept DATA

AXI Interface: Handshaking

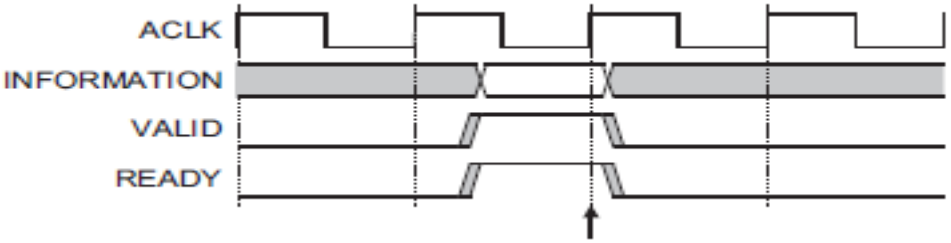
- **AXI uses a valid/ready handshake acknowledge**
- **Each channel has its own valid/ready**
  - Address (read/write)
  - Data (read/write)
  - Response (write only)
- **Flexible signaling functionality**
  - Inserting wait states
  - Always ready
  - Same cycle acknowledge



Inserting Wait States



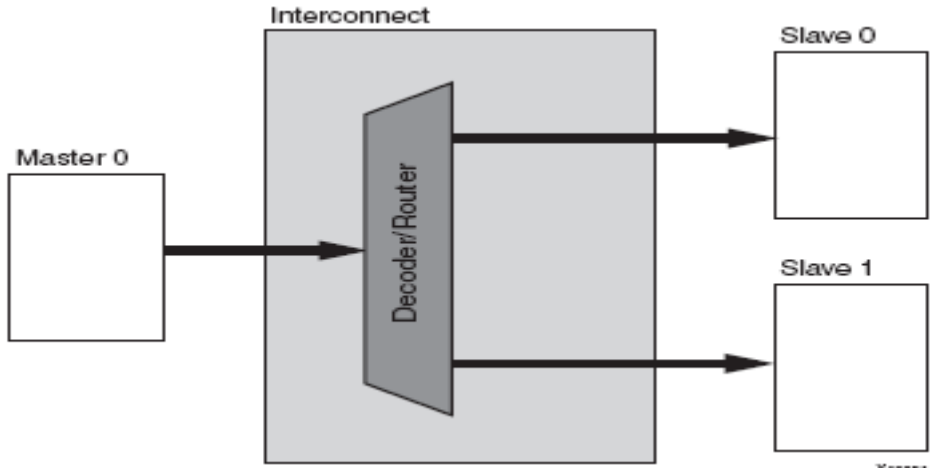
Always Ready

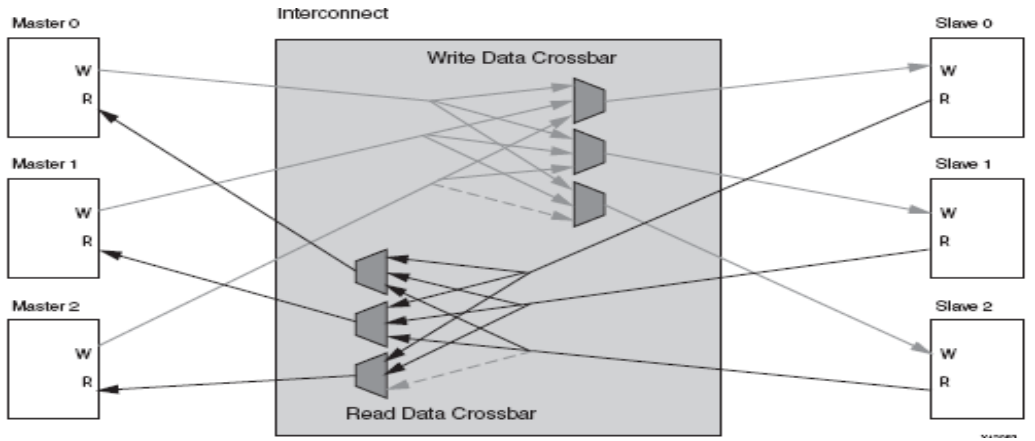


Same Cycle Acknowledge

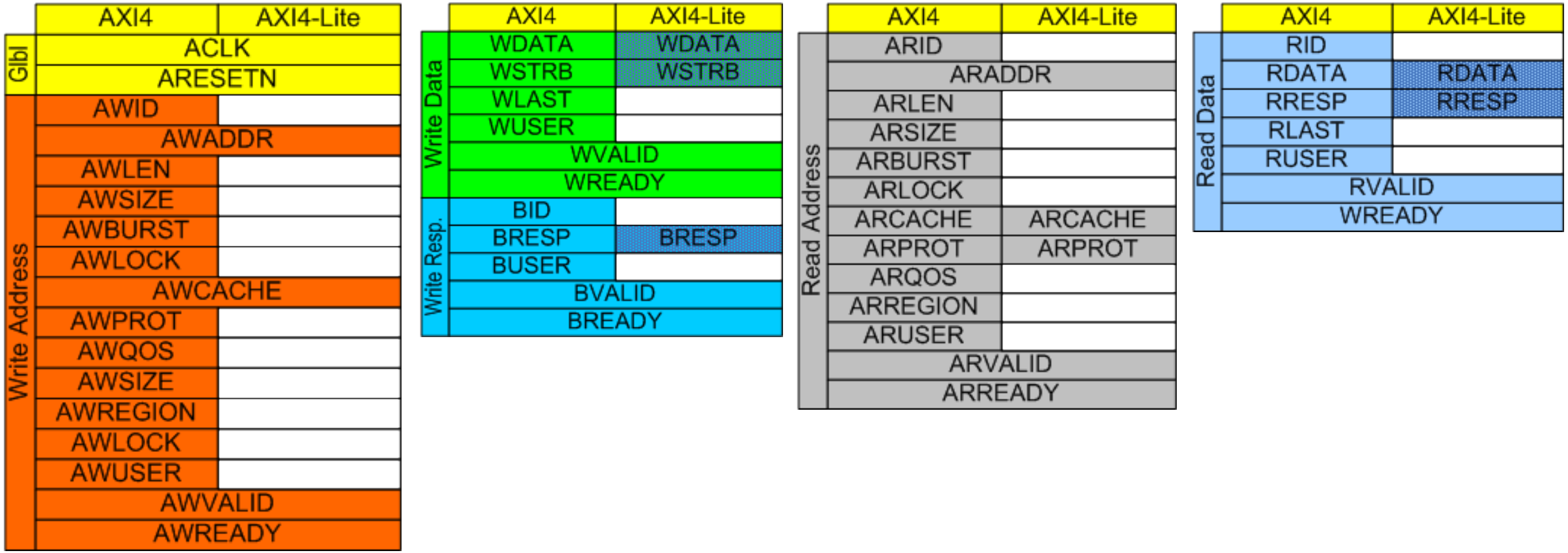
AXI Interconnect

- **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**
- **Xilinx provides three configurable IPIC**
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - AXI4 Slave Burst
- **Xilinx AXI Reference Guide(UG761)**





AXI4 Signals (AXI4, AXI4-Lite)

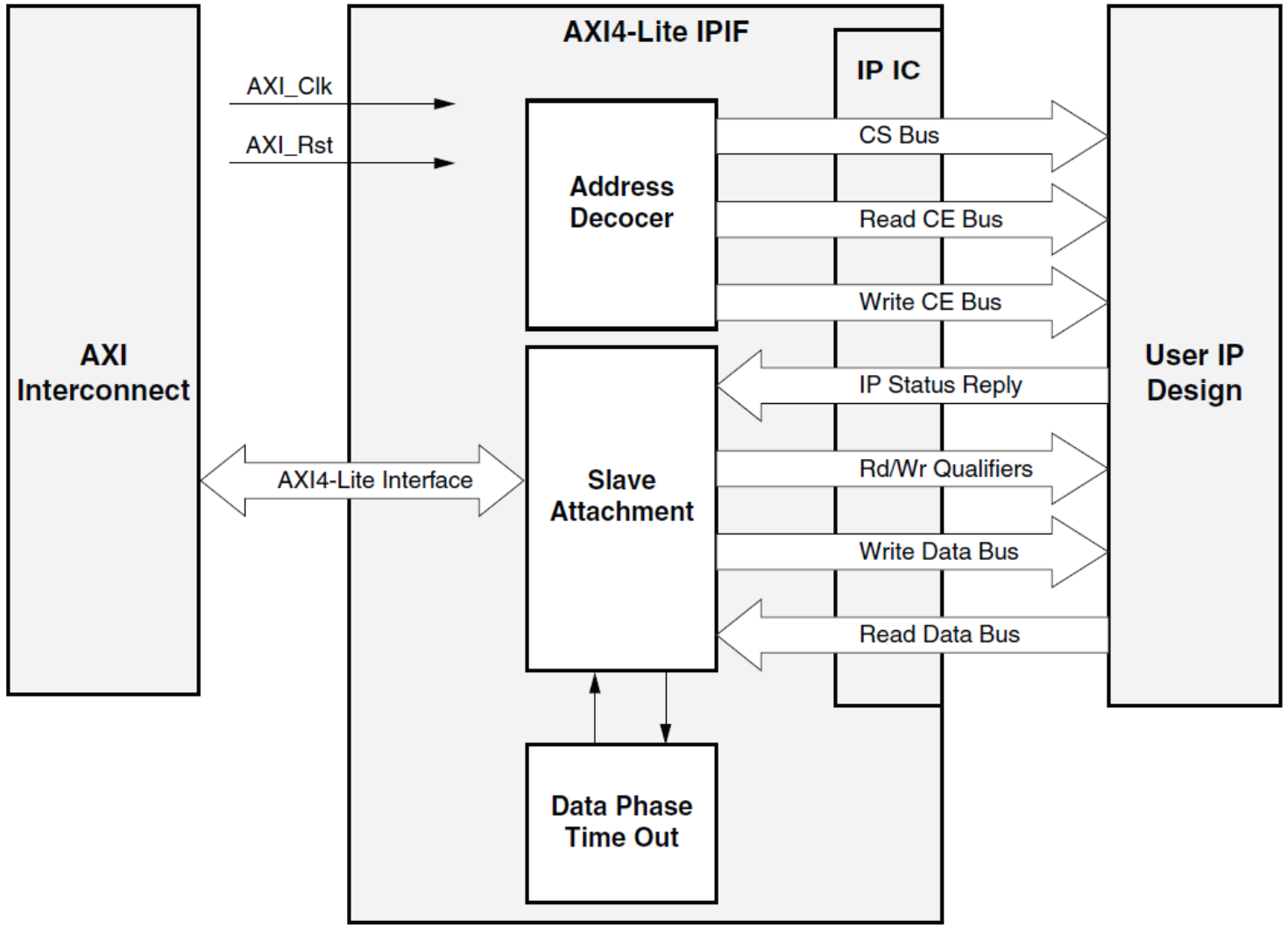


Outline

- ***AXI4 Transactions***
  - *AXI4 Lite Slave*
  - AXI4 Lite Master
  - AXI4 Slave
  - AXI4 Master
- **IP Packager**
- **Custom IP**
- **Summary**

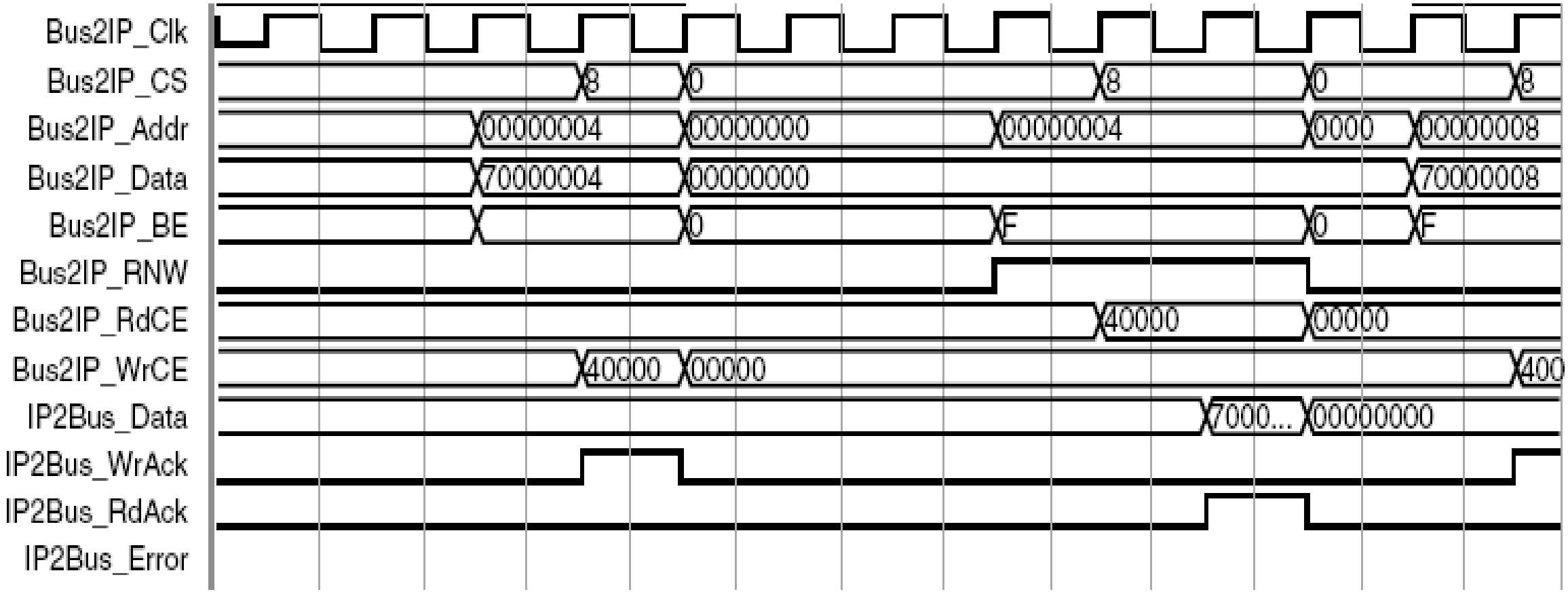
AXI Lite IPIF – Block Diagram

- **Basic services**
  - Slave attachment
  - Address decoding
  - Timeout generation
  - Byte strobe forwarding
- **Optional services**
  - Master user logic
  - Soft reset core
  - User logic software registers, and
  - Timeout logic inclusion



AXI Lite IPIC

Single Data Phase Write and Read Cycle

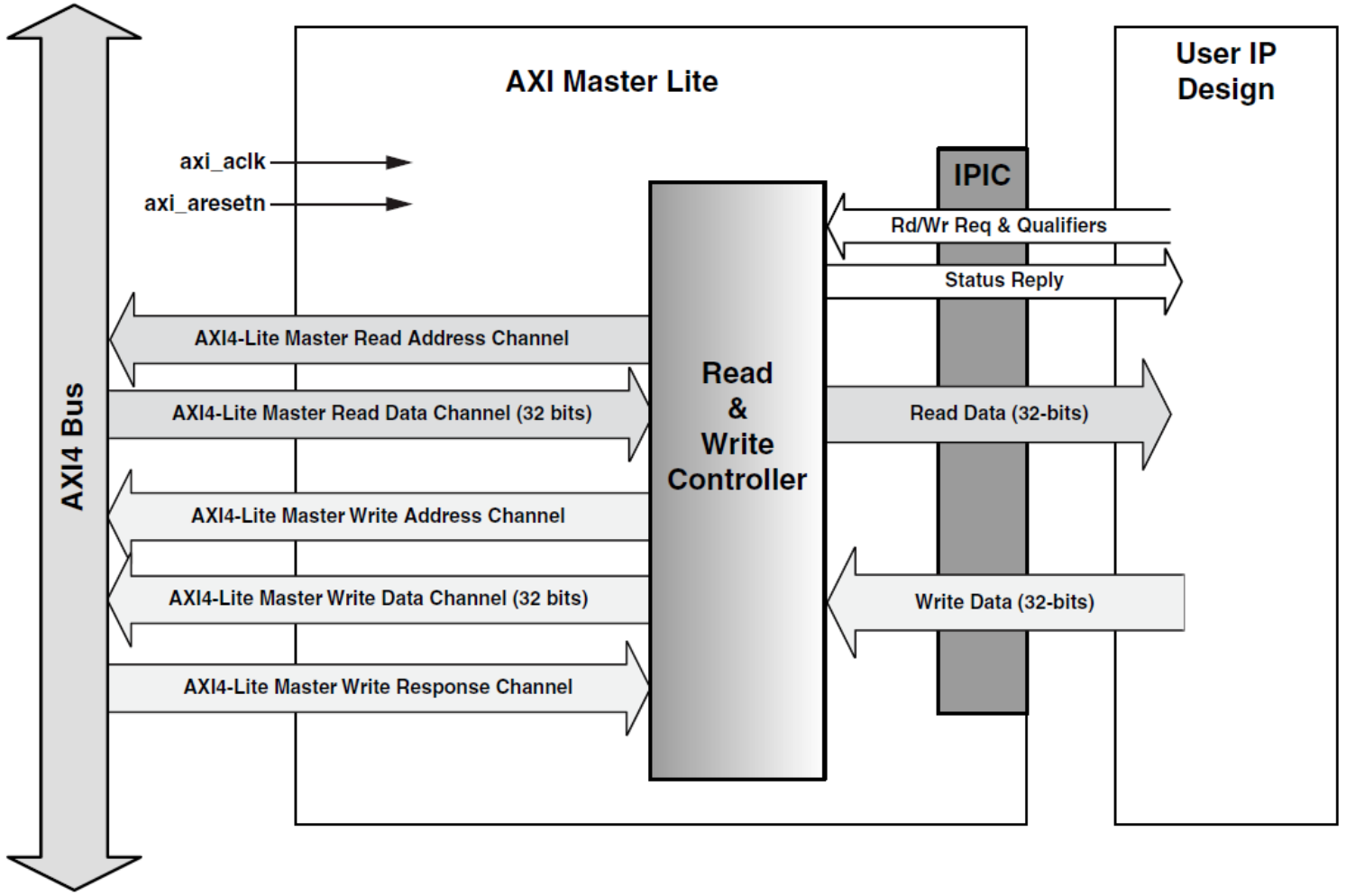


Outline

- ***AXI4 Transactions***
  - AXI4 Lite Slave
  - *AXI4 Lite Master*
  - AXI4 Slave
  - AXI4 Master
- **IP Packager**
- **Custom IP**
- **Summary**

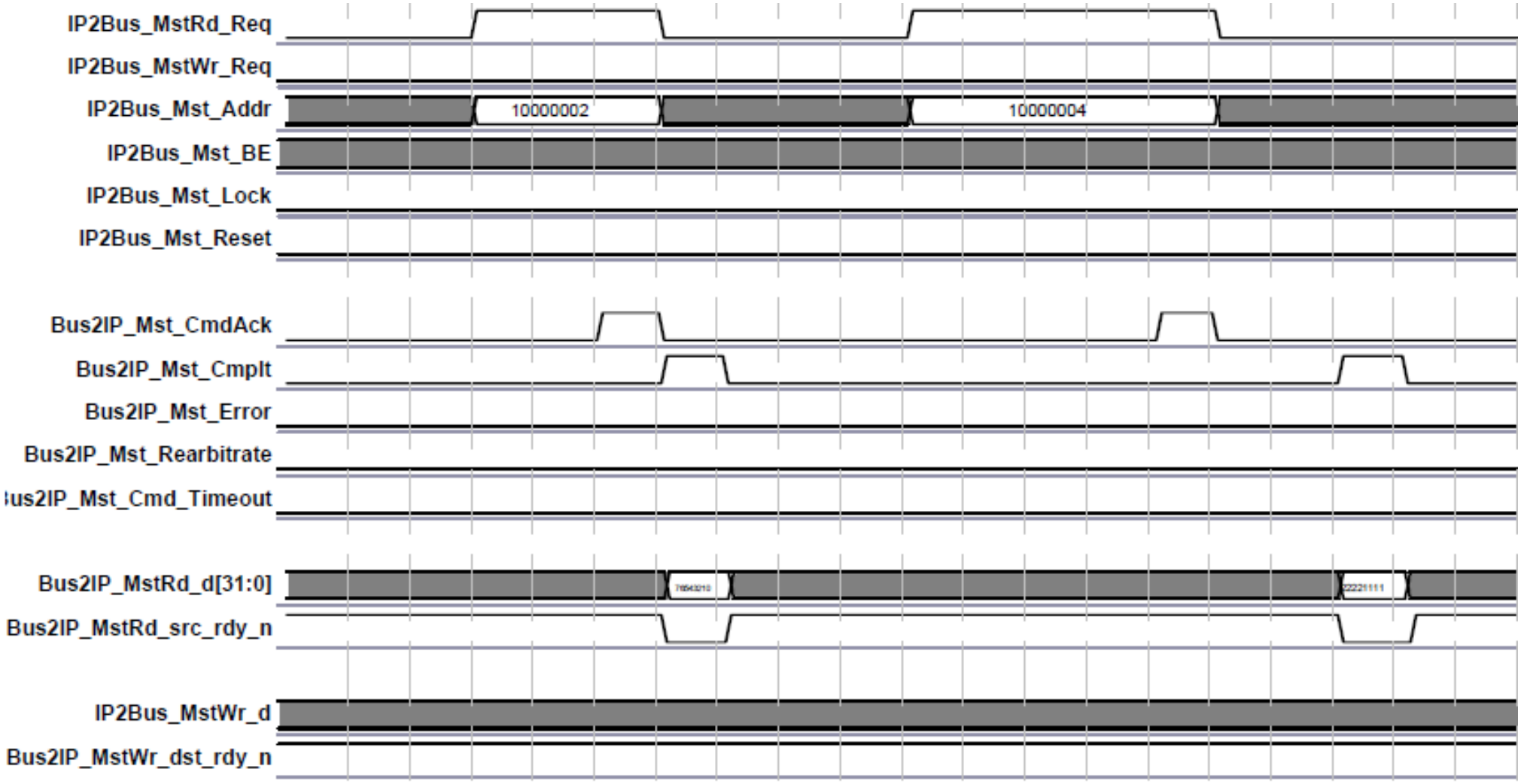
AXI4 Lite Master – Block Diagram

- **AXI4 Lite mastering capability**
- **Single data phase only**
  - One to four bytes
- **Only 32-bit data width**



AXI4 Lite Master

Single Data Phase Read Cycle



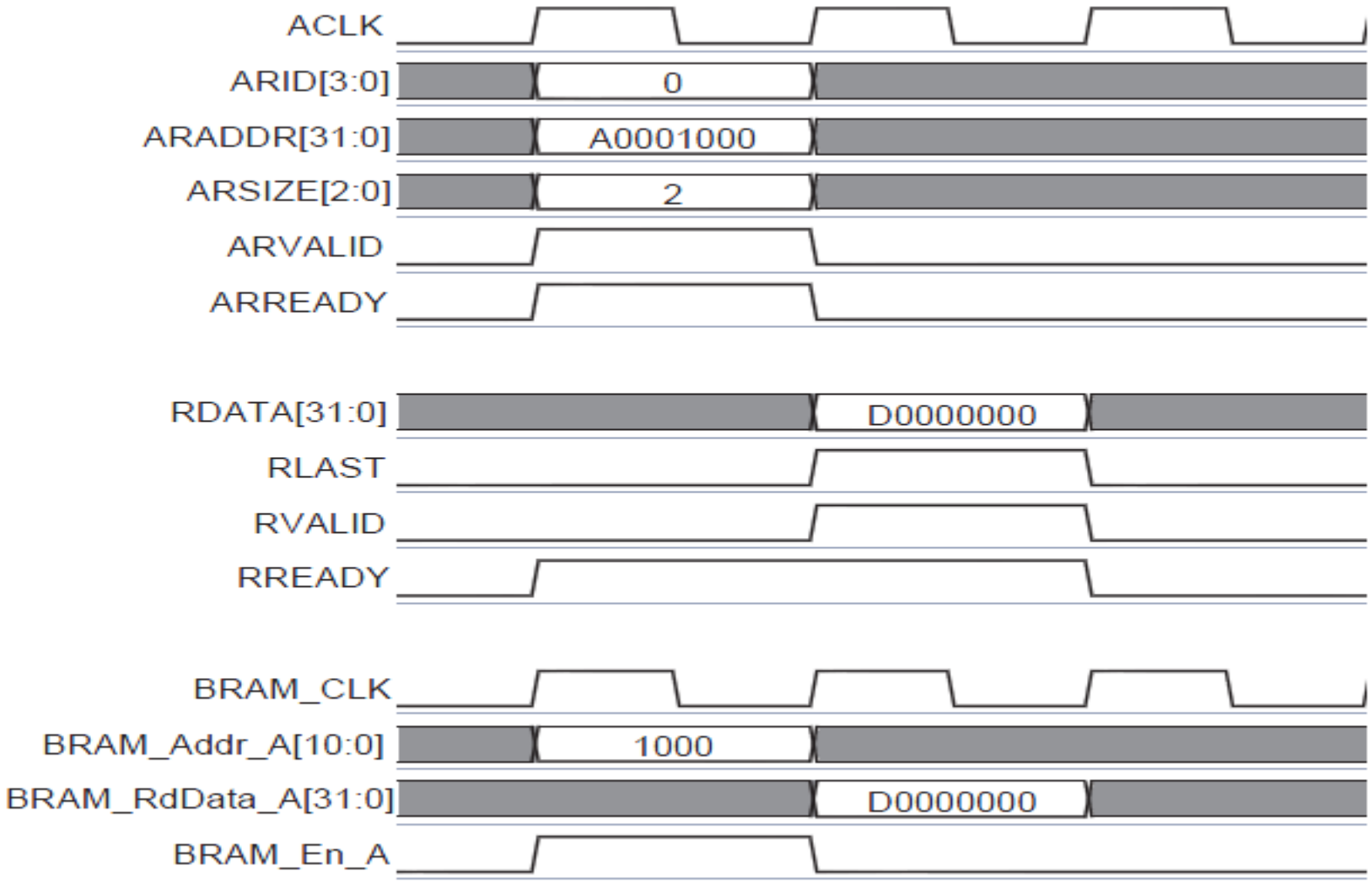
Outline

- ***AXI4 Transactions***
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - *AXI4 Slave*
  - AXI4 Master
- **IP Packager**
- **Custom IP**
- **Summary**

AXI4 Slave
Duties, Options, and Configuration

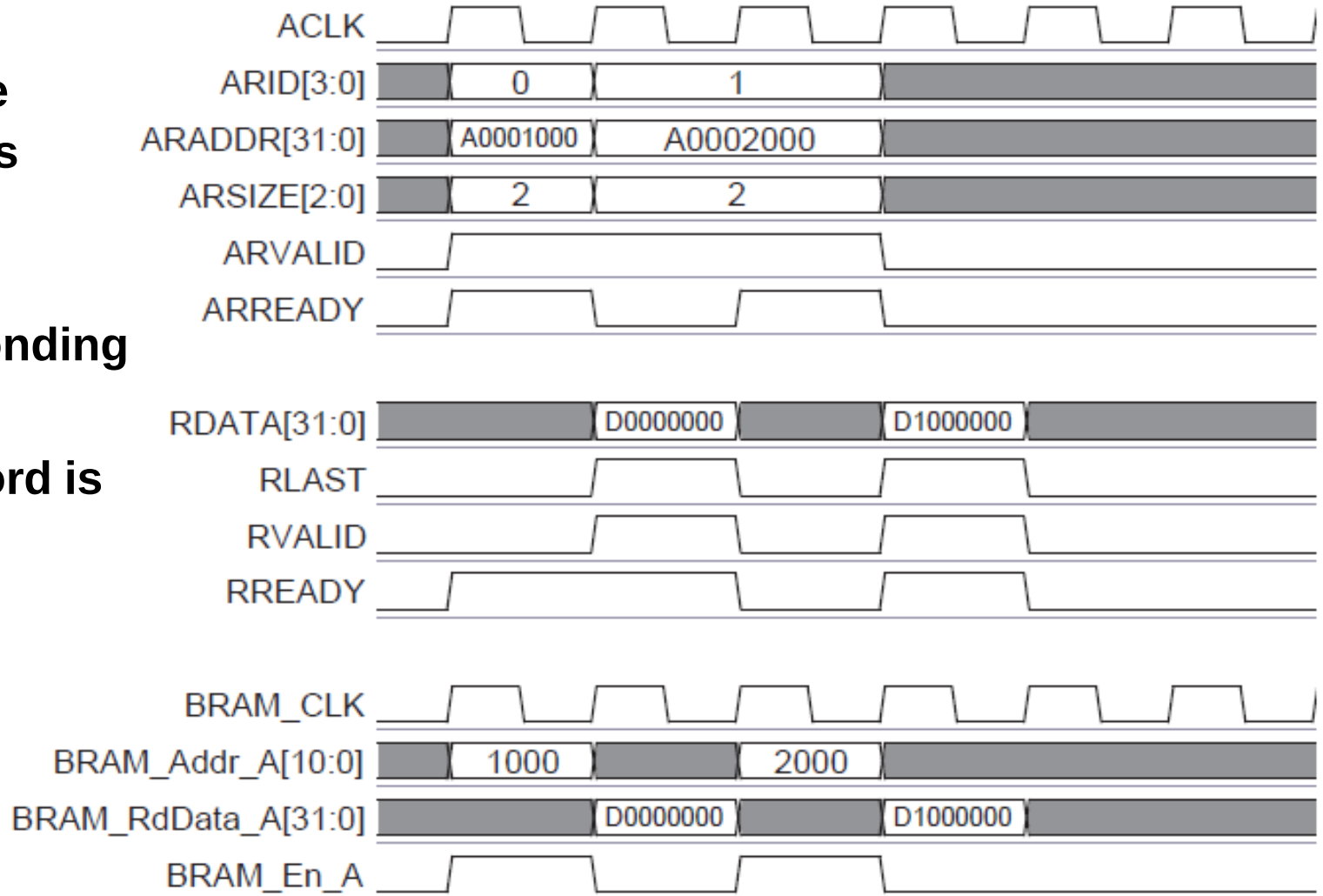
- **Slave IPIC duties and configurable options**
  - Address decode and acknowledge
  - One or more address spaces
    - Memory interface; that is, chip enable
    - User registers
  - Single or burst data phase acknowledgement
  - Software reset/MIR register
  - Read FIFOs
  - Automatic timeout on user slave logic
  - Your custom slave attachments

AXI4 Read Transaction

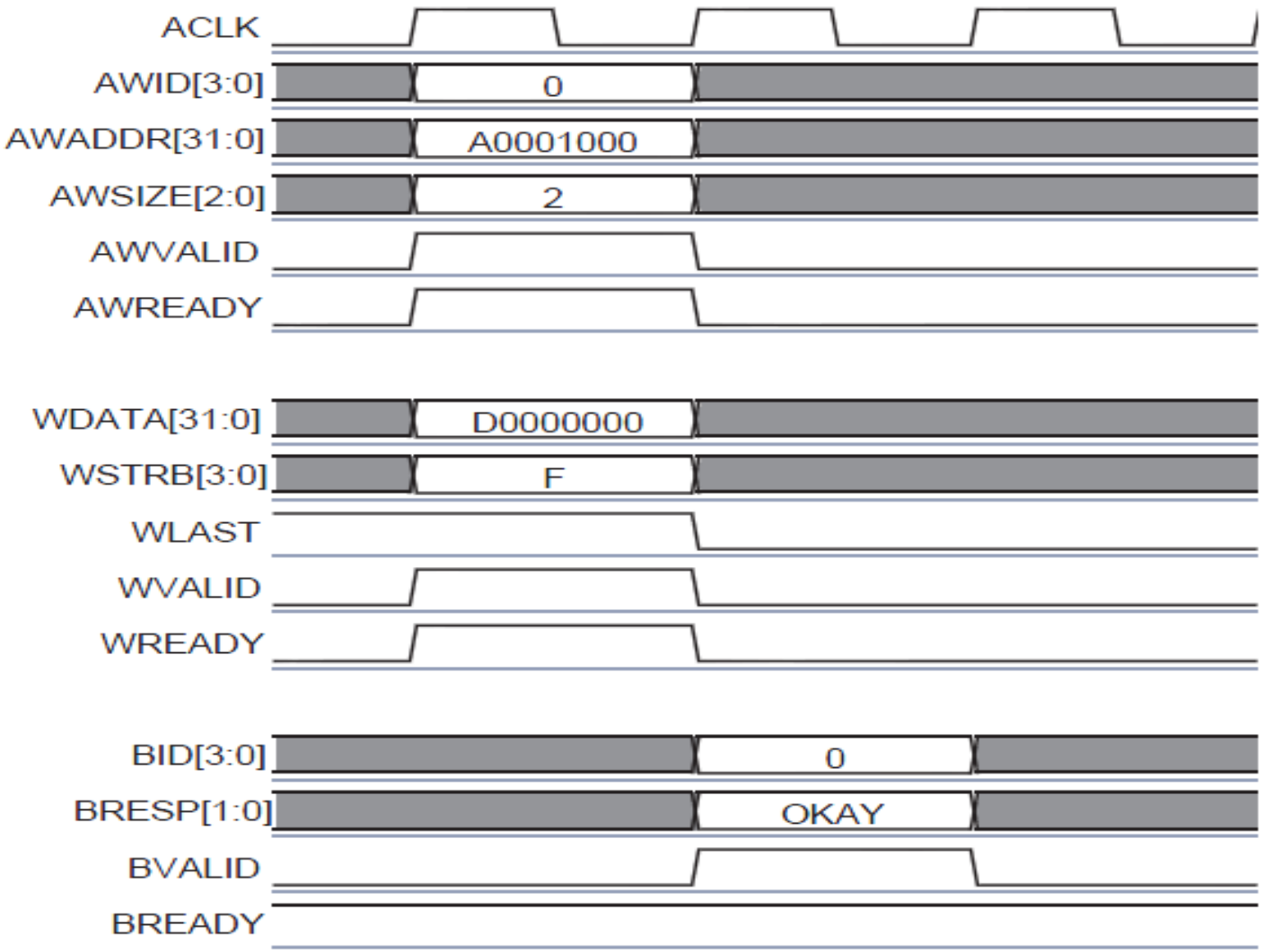


AXI4 Read Transaction
Multiple reads, not burst transaction

- **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 entire word is being read**

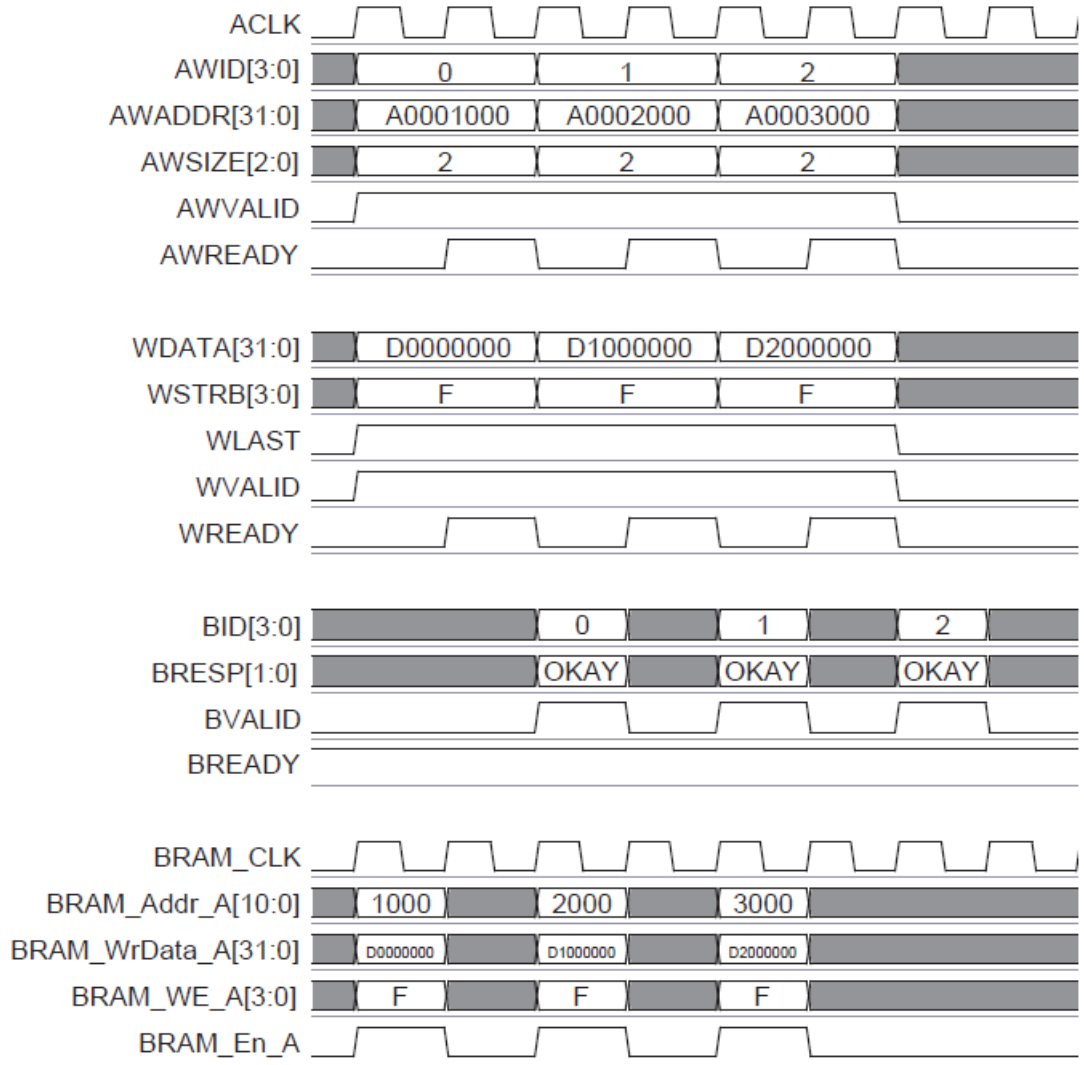


AXI4 Write Transaction

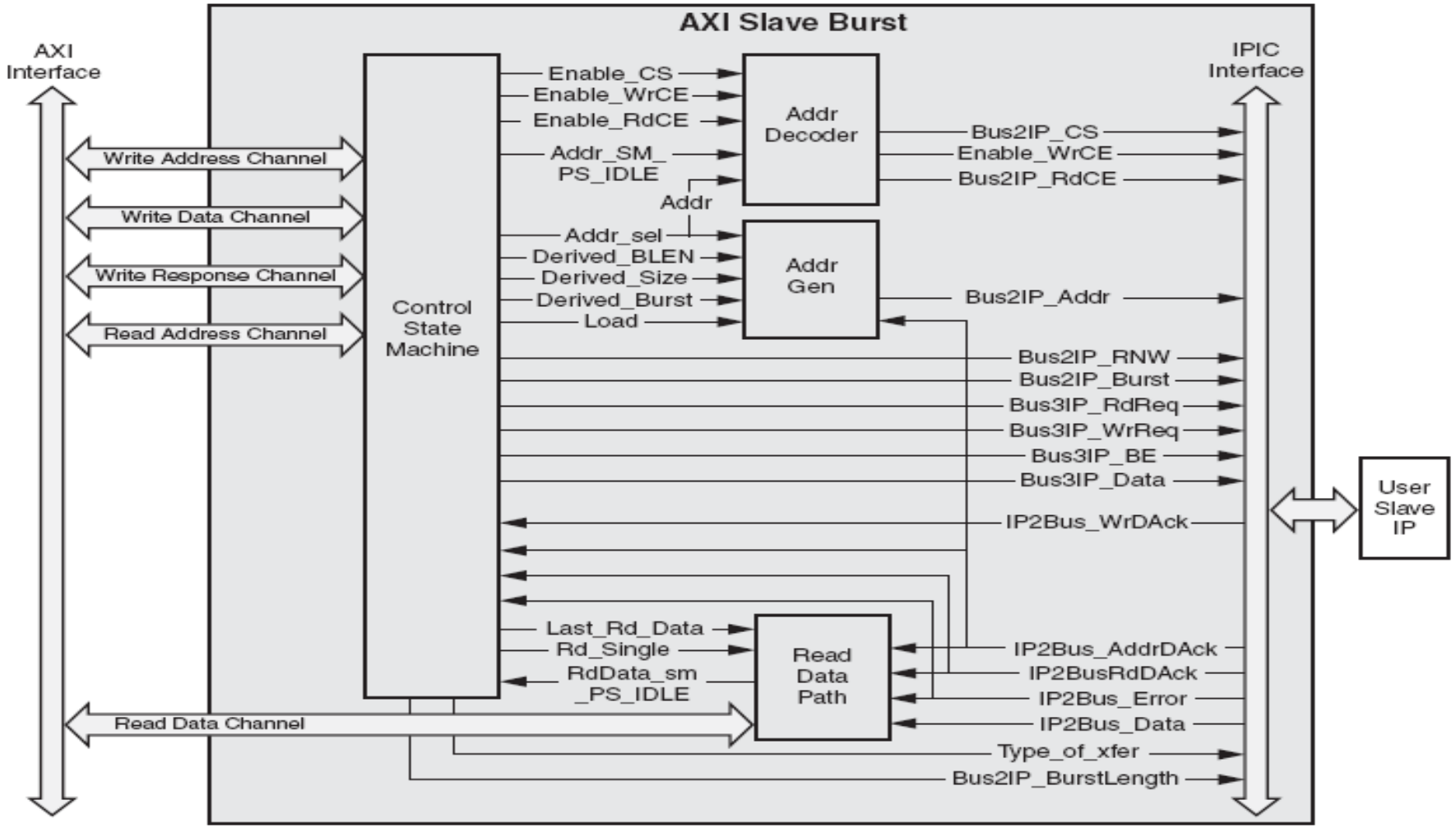


AXI4 Write Transaction
Multiple write, not burst

- **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**

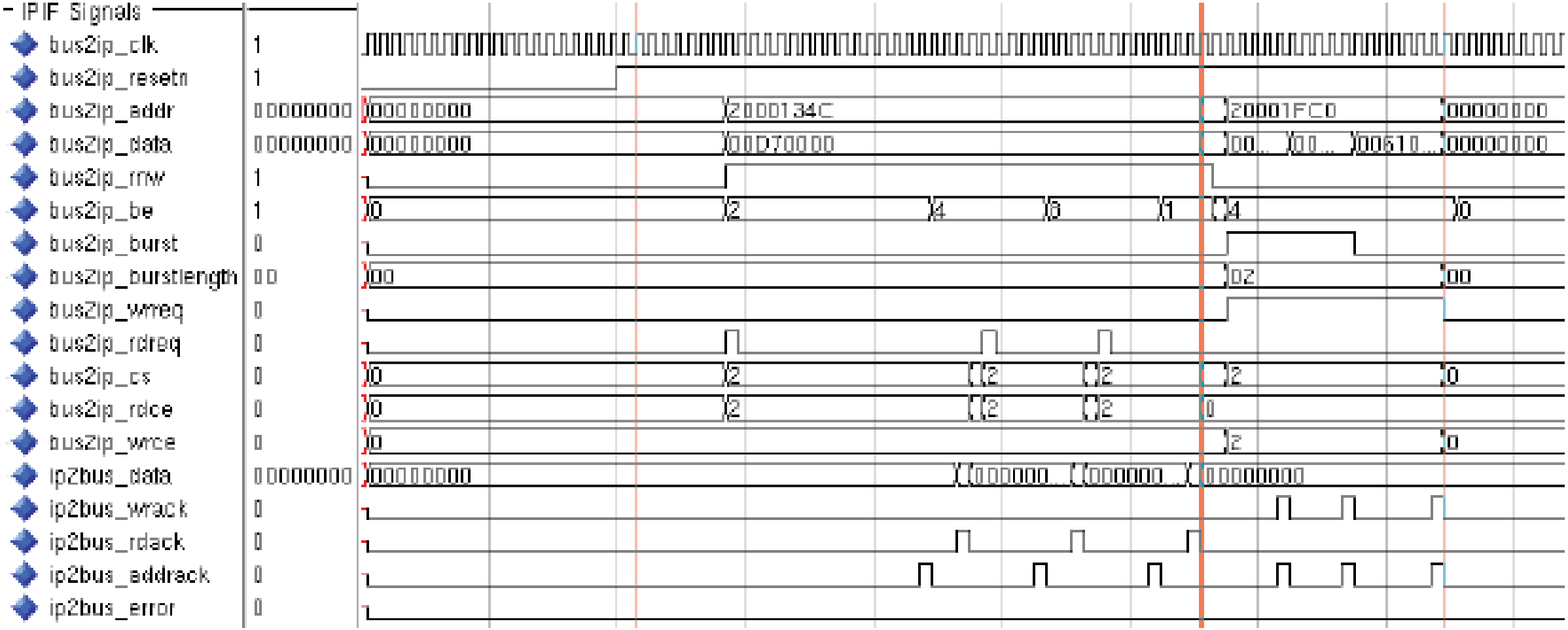


AXI Slave Burst – Block Diagram



AXI Slave Burst

Burst Data Phase 3 Reads and 3 Writes

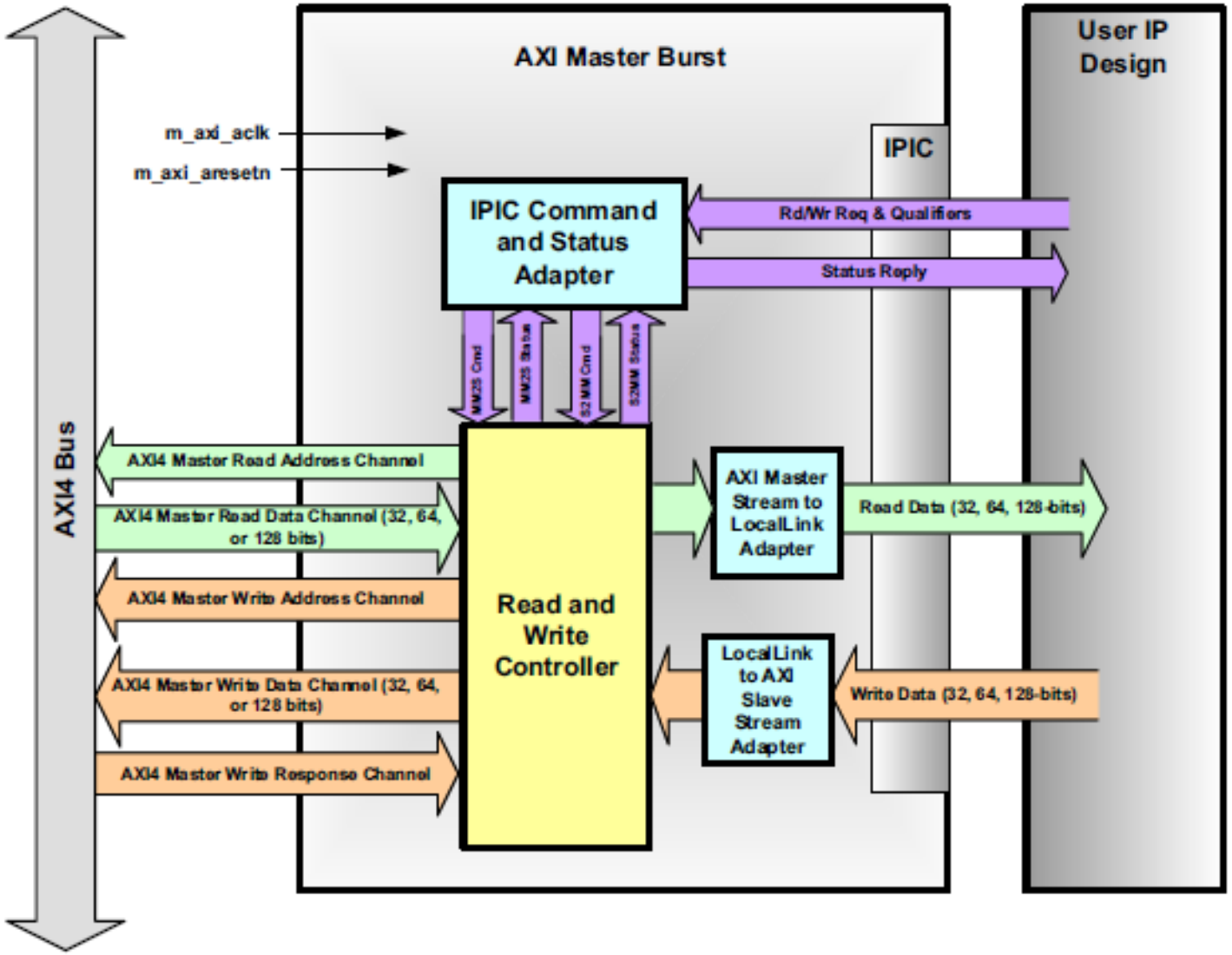


Outline

- ***AXI4 Transactions***
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - AXI4 Slave
  - *AXI4 Master*
- **IP Packager**
- **Custom IP**
- **Summary**

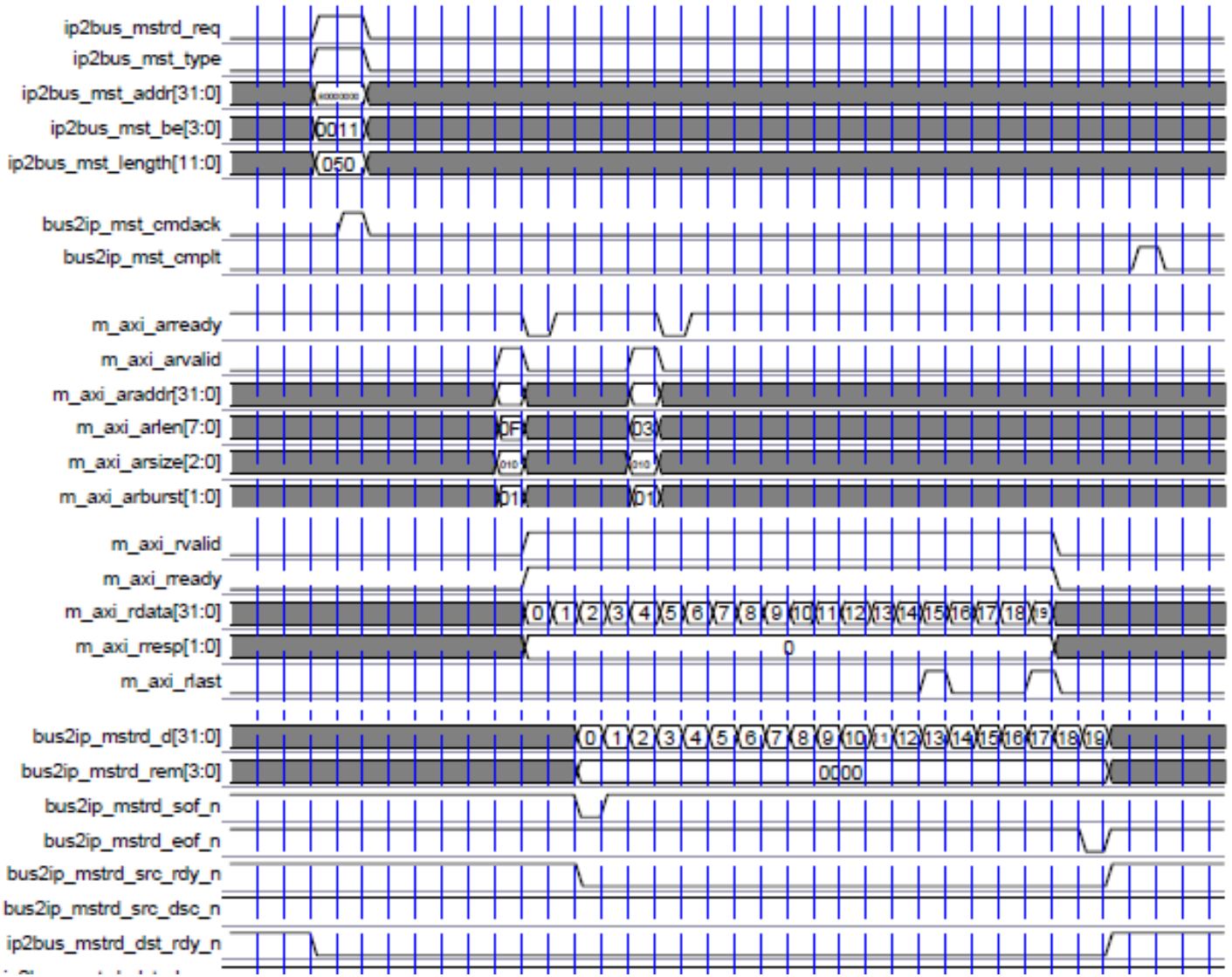
AXI4 Master Burst – Block Diagram

- **Parameterizable data width**
  - 32, 64, 128
- **Data burst**
  - 16, 32, 64, 128, 256 data beats



AXI4 Master Burst Read

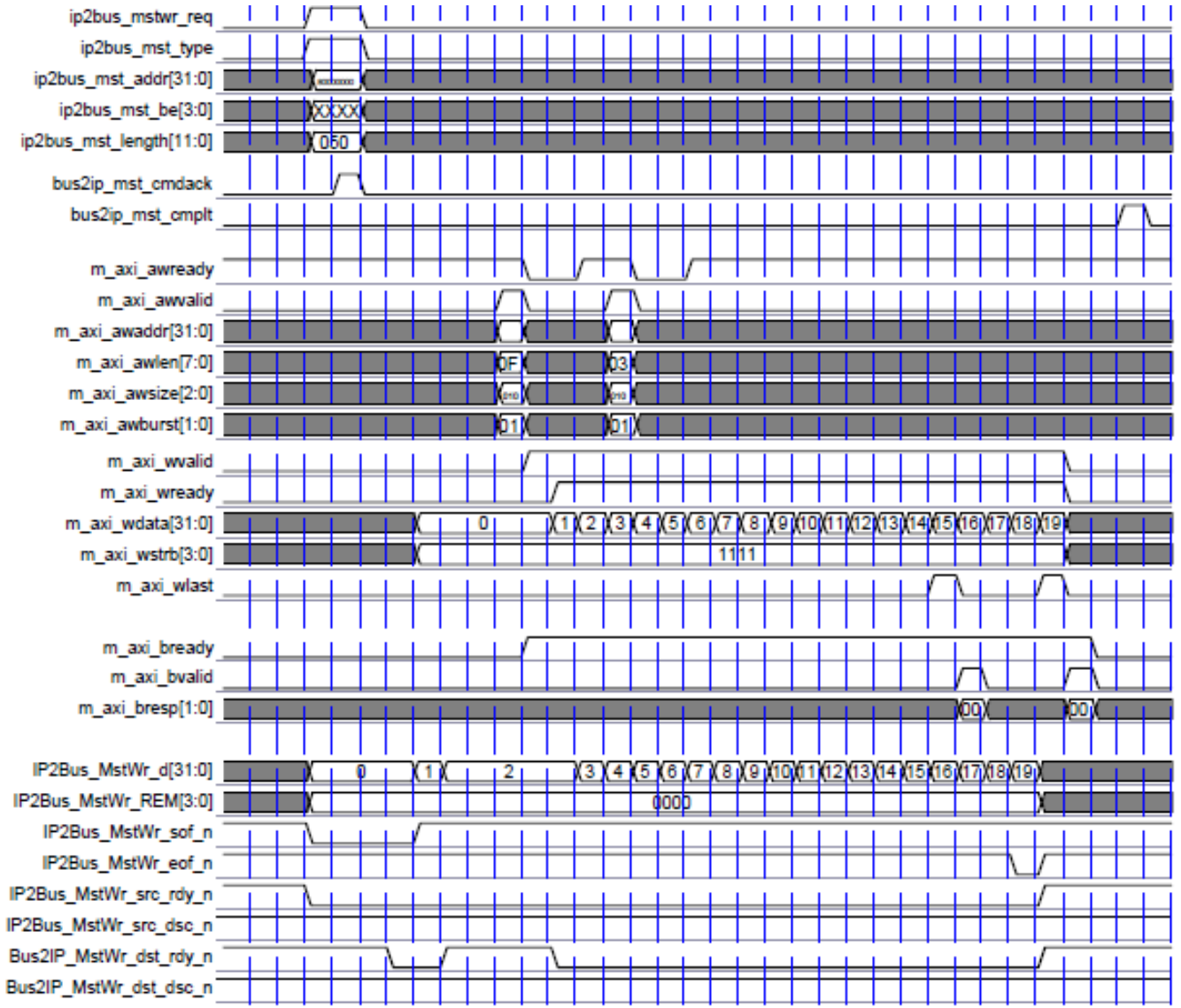
- **80 bytes of burst read**
  - ip2bus_mst_length signal
- **Transaction delimited by**
  - bus2ip_mst_cmdack and bus2ip_mst_cmplt signals
- **Burst read broken into two transactions**
  - 16 data beats (64 bytes)
  - 4 data beats (16 bytes)
- **AXI master receive data**
  - m_axi_rdata
- **User logic receive data**
  - bus2ip_mstrd_d
- **Data framing**
  - bus2ip_mstrd_sof_n
  - bus2ip_mstrd_eof_n



Only relevant signals are shown for readability purpose

AXI4 Master Burst Write

- **80 bytes of burst write**
  - ip2bus_mst_length signal
- **Transaction delimited by**
  - bus2ip_mst_cmdack and bus2ip_mst_cmplt signals
- **Burst write broken into two transactions**
  - 16 data beats (64 bytes)
  - 4 data beats (16 bytes)
- **User logic writes data**
  - IP2Bus_MstWr_d
- **AXI master write data**
  - m_axi_wdata
- **Data framing**
  - IP2Bus_MstWr_sof_n
  - IP2Bus_MstWr_eof_n



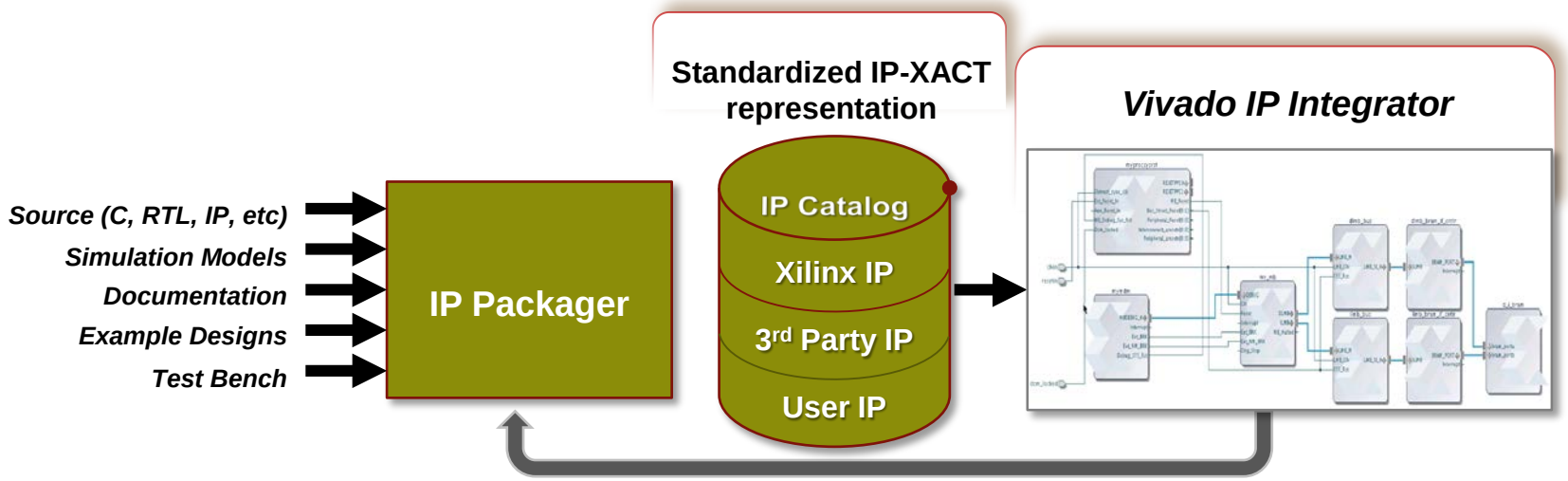
Only relevant signals are shown for readability purpose

Outline

- **AXI4 Transactions**
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - AXI4 Slave
  - AXI4 Master
- ***IP Packager***
- **Custom IP**
- **Summary**

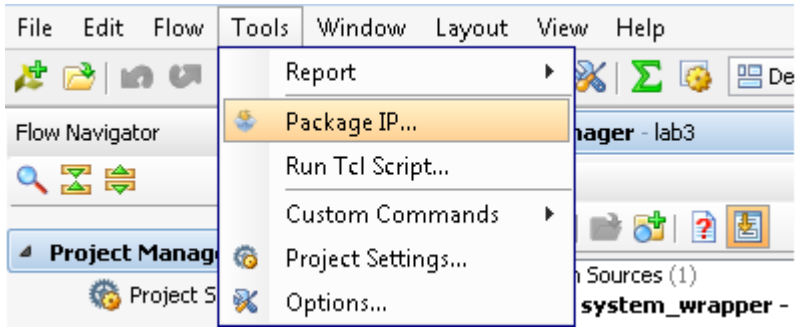
Reusing Your IP

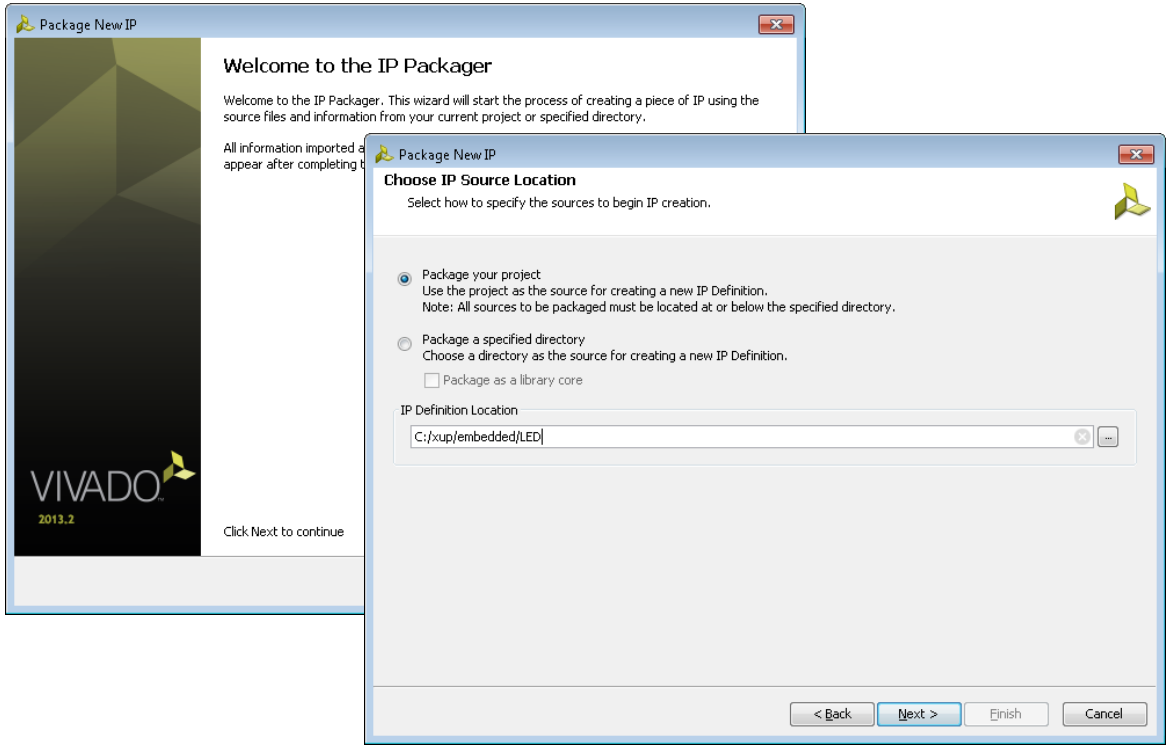
- IP from many sources can be packaged and made available in Vivado
- All IP available in the Vivado IP Catalog can be used to create IP Integrator designs
- Any IP Integrator diagram can be quickly packaged as a single complex IP



IP Packager

- **The IP Packager allows a core to be packaged and included in the IP Catalog, or for distribution**
- **IP-XACT**
- **Complete set of files include**
  - Source code, Constraints, Test Benches (simulation files), documentation
- **IP Packager can be run from Vivado on the current project, or on a specified directory**



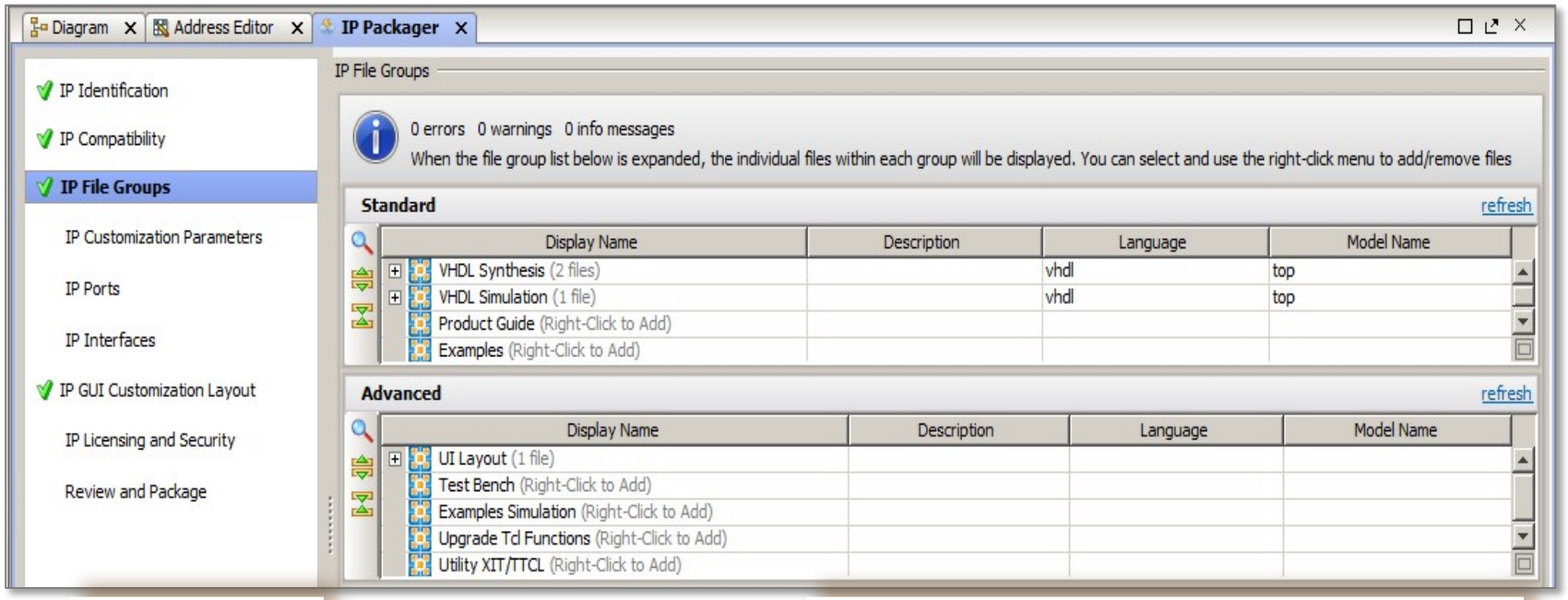


IP-XACT

- **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)

Customizing IP for Reuse in IP Packager

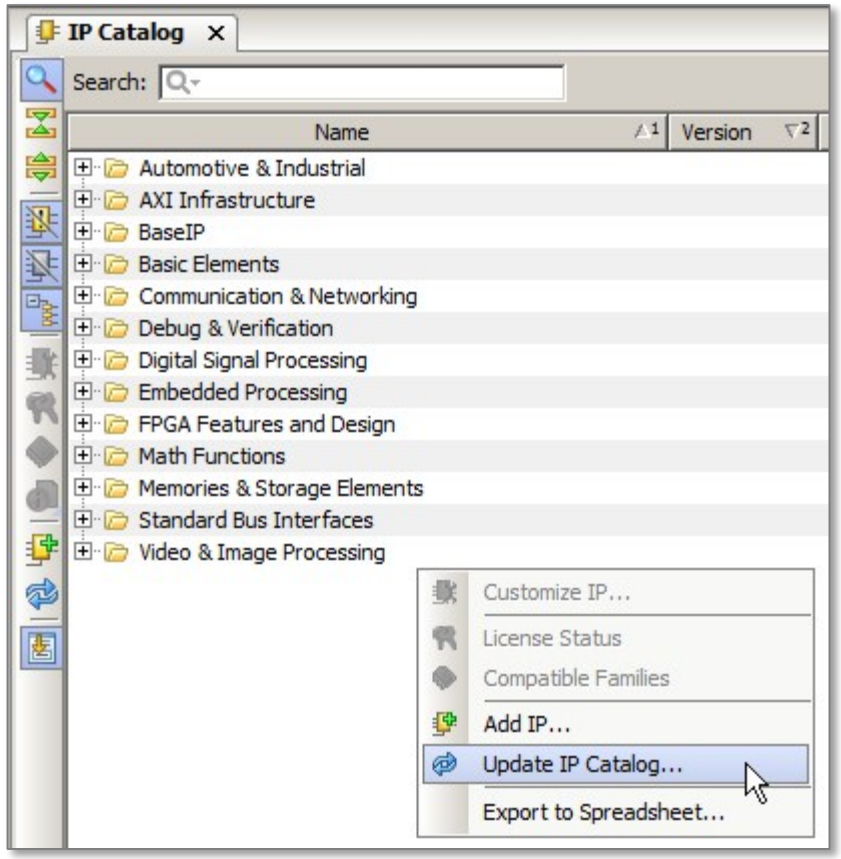


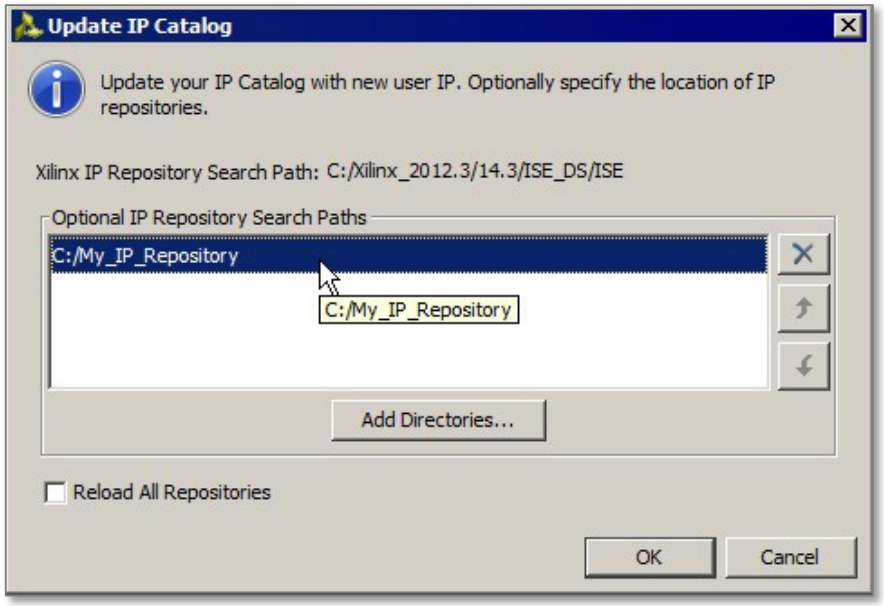
Select Options

Add, Edit or change defaults

Using and Reusing Packaged IP

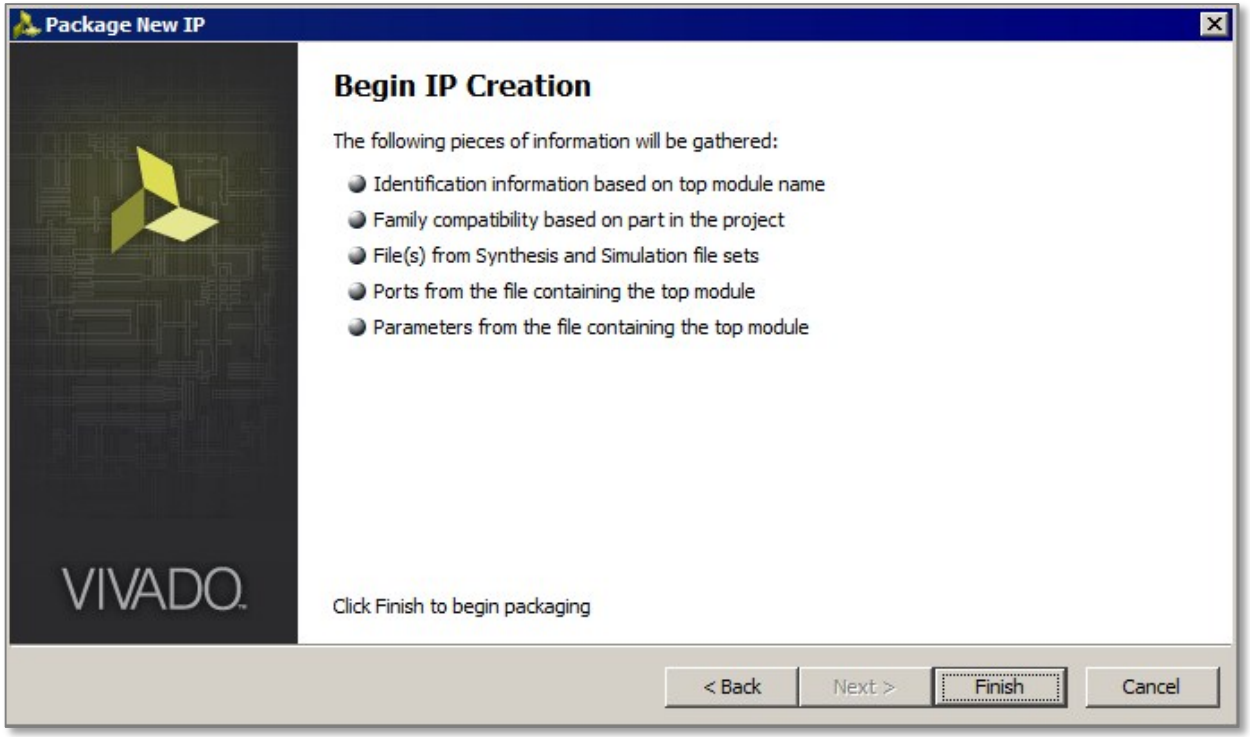
- The Vivado IP Catalog can be extended by adding additional IP Repositories. Third party IP, your custom IP, and Xilinx provided IP are displayed in an identical manner





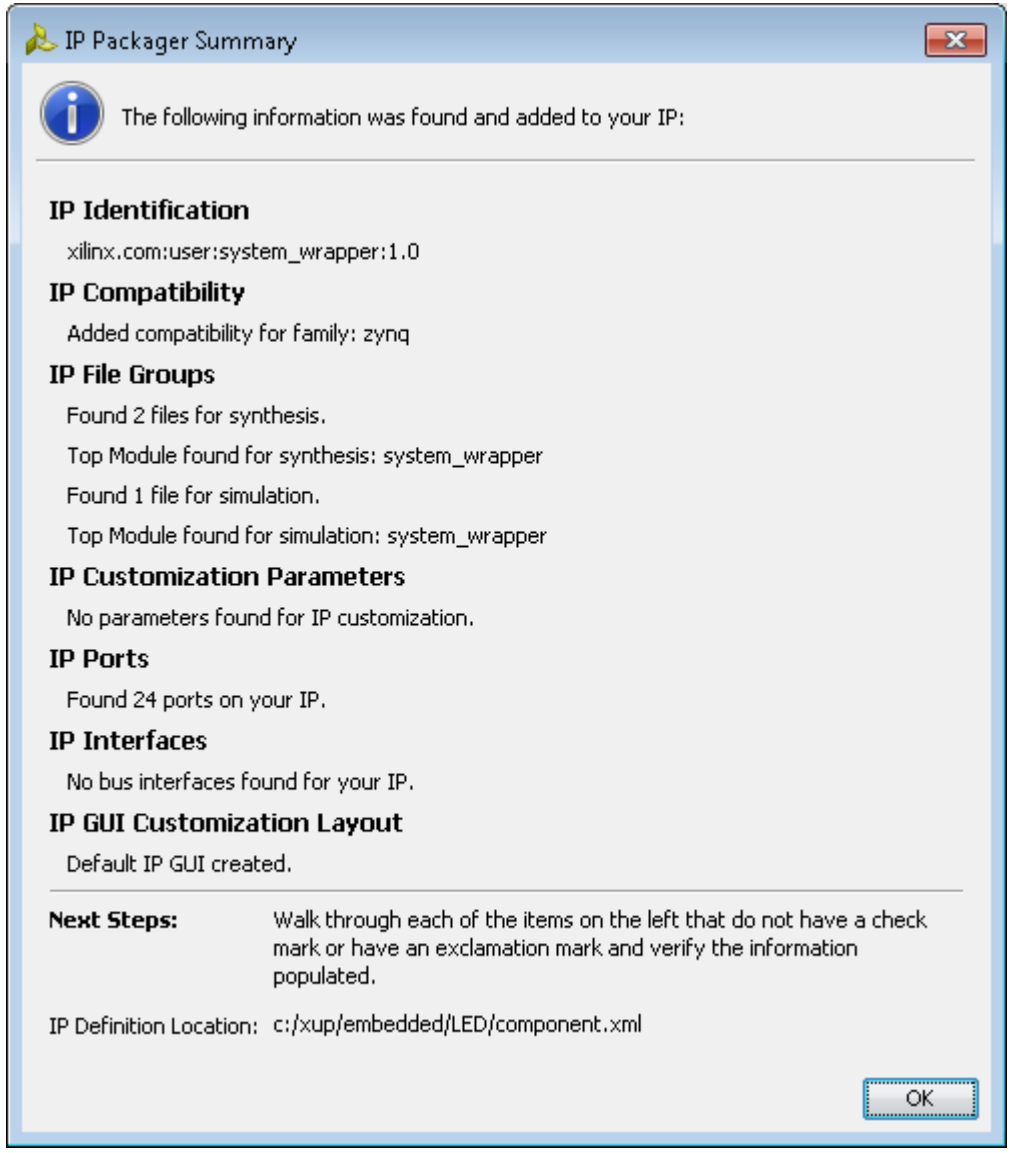
Capture Your IP Using the Vivado IP Packager

- **Wizard-based flow**
  - Automates generation of IP-XACT IP
  - Many pieces of meta-data automatically inferred
  - Users can add additional meta-data



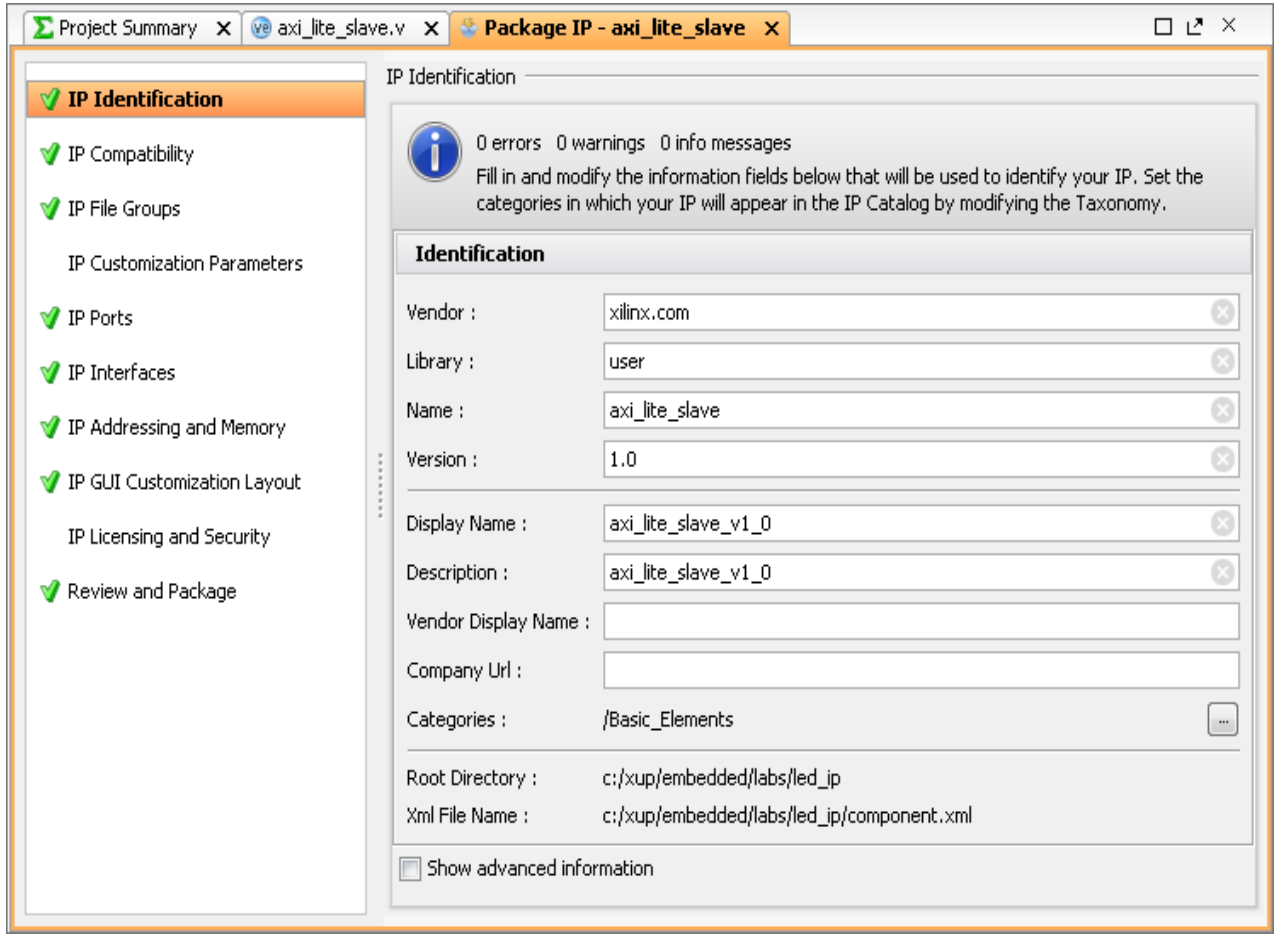
IP Packager

- **Automatically analyze project/files to determine parameters**
- **Initial Summary**
- **Identifies**
  - Files
    - Source HDL, Testbenchs, Documentation,
  - Parameters
    - Configurable
  - Ports
  - Interfaces
  - Compatability
- **Creates GUI Layout for IPI**



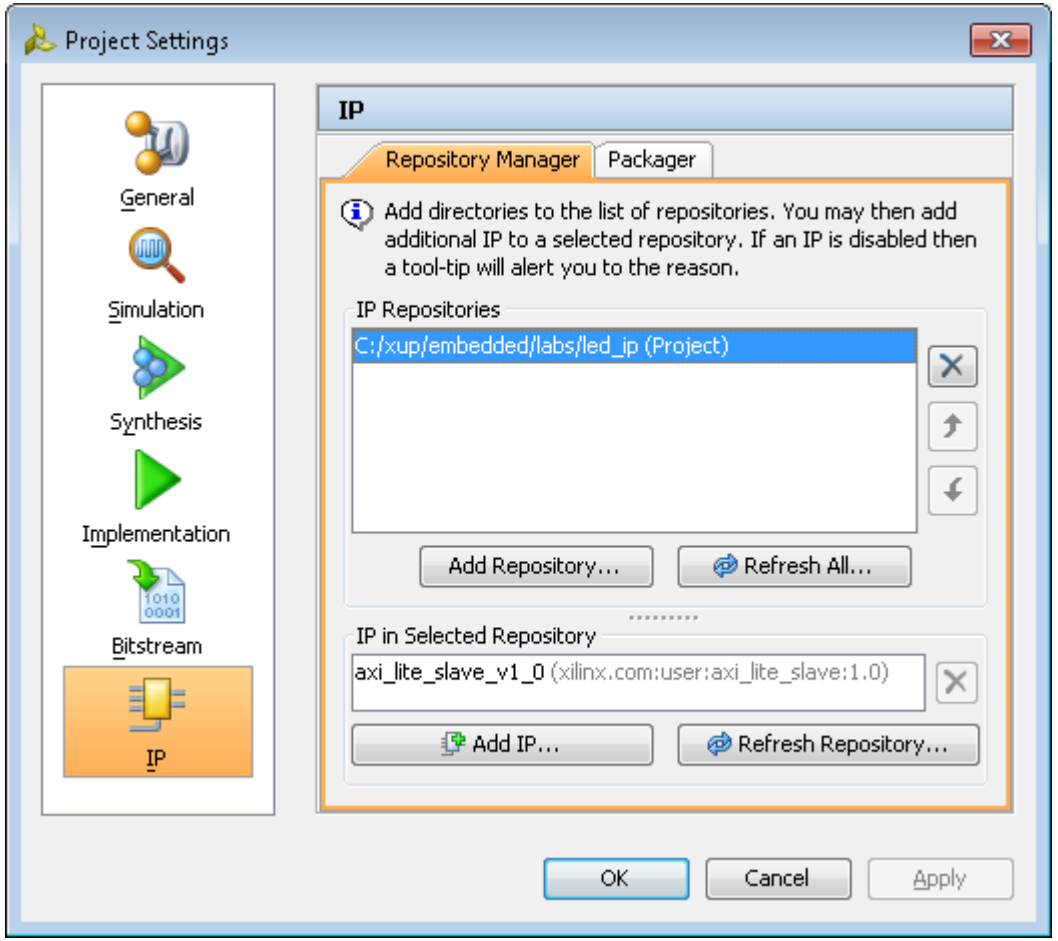
IP Packager

- **Modify configuration**
  - Properties
  - Compatibility
  - Files
  - Custom parameters
  - Ports
  - Interfaces
  - Address and Memory
  - IP and security



IP repository

- Creates .xml file for the IP
- Specify the directory in the repository
- Displays IP in the repository

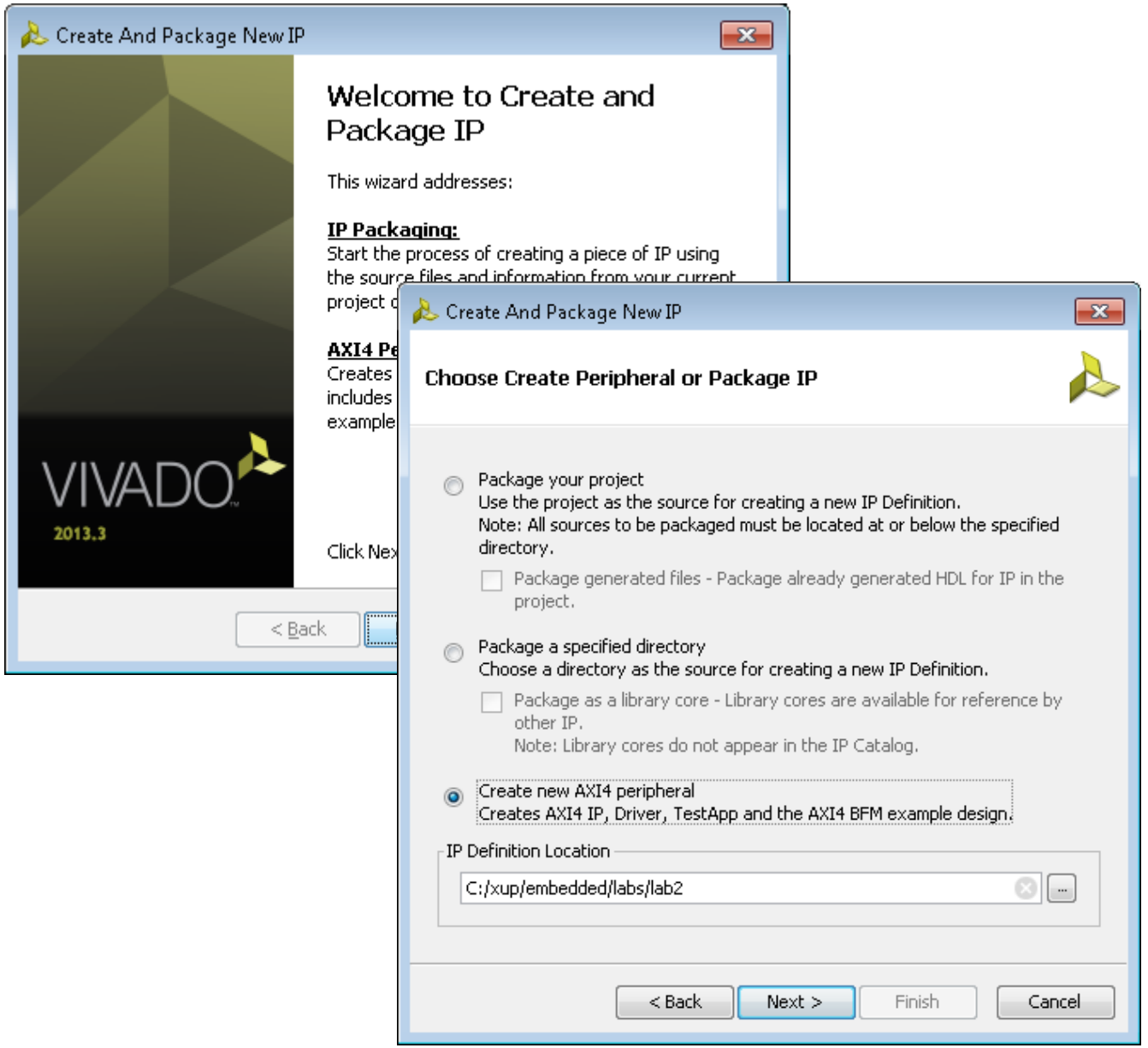


Outline

- AXI4 Transactions
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - AXI4 Slave
  - AXI4 Master
- IP Packager
- *Custom IP*
- Summary

Create Custom IP (Starting in 2013.3)

- **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



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_inc
      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_inc
      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_inc
      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_inc
      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
```

Address

Register

Data

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 source
- **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;
      }
```

Using the IP

- **Modify template, add user logic**
- **Package IP**
  - Specify configuration settings
- **Specify IP directory in the Vivado repository**
  - Project settings
- **Use in IP Catalog like any other IP**

Outline

- AXI4 Transactions
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - AXI4 Slave
  - AXI4 Master
- IP Packager
- 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 Import wizard supports AXI Lite, Full, and Stream compatible IP creation**
  - Handles interface side protocol
  - Provides template to add HDL functionality