ALL PROGRAMMABLE

Creating and Adding Custom IP

Zynq Vivado 2013.3 Version

This material exempt per Department of Commerce license exception TSU

© Copyright 2013 Xilinx

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



Slave 0

Slave 1

Slave 0

Slave 1

Slave 2

w

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)



Interconnect

Interconnect

Decoder/Router

Write Data Crossbar

Read Data Crossbar

Master 0

Master 0

Master 1

Master 2

AXI4 Signals (AXI4, AXI4-Lite)

	AXI4	AXI4-Lite							
p	AC	LK							
G	ARES	SETN							
	AWID								
	AWA	DDR							
	AWLEN								
	AWSIZE								
	AWBURST								
SSS	AWLOCK								
dre	AWCACHE								
Ad	AWPROT								
ite	AWQOS								
Ň	AWSIZE								
	AWREGION								
	AWLOCK								
	AWUSER								
	AWV	ALID							
	AWR	EADY							

AXI4	AXI4-Lite
WDATA	WDATA
WSTRB	WSTRB
WLAST	
WUSER	
WVA	ALID
WRE	ADY
BID	
BRESP	BRESP
BUSER	
BVA	LID
BRE	ADY

	AXI4	AXI4-Lite
	ARID	
	ARA	DDR
	ARLEN	
	ARSIZE	
SSS	ARBURST	
dr	ARLOCK	
Pq	ARCACHE	ARCACHE
ad	ARPROT	ARPROT
е К	ARQOS	
	ARREGION	
	ARUSER	
	ARV	ALID
	ARRE	EADY

	AXI4	AXI4-Lite							
	RID								
B	RDATA	RDATA							
at	RRESP	RRESP							
뉭	RLAST								
ea	RUSER								
œ ا	RVÁLID								
	WREADY								

XILINX ➤ ALL PROGRAMMABLE.

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

Bus2IP_Clk											
Bus2IP_CS			_X≊				X 8		_x_		X®_
Bus2IP_Addr) 000	00004	 000	00000) 0000	00004) 000	000	80000
Bus2IP_Data) 700	00004) (000	00000) 700	80000
Bus2IP_BE				X_		_XF			X0	_XE	
Bus2IP_RNW											
Bus2IP_RdCE							/ 4000	0	000	00	
Bus2IP_WrCE			X 400	00 X 000	00						X 400
IP2Bus_Data) 700	0 X 000	00000	
IP2Bus_WrAck											
ID2Due DdAck											

F

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

	ACLK								
ļ	ARID[3:0]	X	0)	1		(
5	ARADDR[31:0]) /	A0001000) A	000200	00	(
	ARSIZE[2:0]	X	2)	2		(
	ARVALID						1		
	ARREADY						1		
nding									
	RDATA[31:0]			D00000	000		D10000	00)	
rd is	RLAST								
	RVALID								
	RREADY								
	BRAM_CLK								
BRA	M_Addr_A[10:0]	X	1000)	(2	2000			
BRAM_	RdData_A[31:0]			D00000	000 (D10000) (00	
	BRAM_En_A								

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

ACLK				
AWID[3:0]	0	1	2	(
AWADDR[31:0]	A0001000	A0002000	A0003000	
AWSIZE[2:0]	2	2	2	
AWVALID				L
AWREADY				L
WDATA[31:0]	D0000000	D1000000	D2000000	(
WSTRB[3:0]	(F	(F) F	
WLAST				L
WVALID				L
WREADY		\		L
BID[3:0]		(0)	1	2
BRESP[1:0]		OKAY	OKAY	(OKAY)
BVALID				
BREADY				
BRAM_CLK				
BRAM_Addr_A[10:0]	(1000)	2000	3000	
BRAM_WrData_A[31:0]	D0000000	D1000000	D2000000	
BRAM_WE_A[3:0]	(F)) F)	(F)	
BRAM_En_A				

AXI Slave Burst – Block Diagram



XILINX > ALL PROGRAMMABLE..

AXI Slave Burst Burst Data Phase 3 Reads and 3 Writes

- IP	IF Signals ————												
+	bus2ip_clk	1	התתתחתת	իորորուներ	תתתחתת	תתתתת	ותת ותות ותתו	תהתחתתה	חחחת	ווווווו		տորո	սերորուրիսու
•	bus2ip_resetr	1											
- 🄶	busZip_addr	0000000	000000000		2	1001 34 C				_]20(101FC0		000000000
- 🏘	busZip_data	0000000	<u>)000000000</u>		<u>)</u> 0	1070000				_100	<u>Xoo</u>	<u>)00610</u>	
- 🔶	bus2ip_rnw	1	1										
-	bus2ip_be	1)0)2)4)B)I	1.14			Xo
- 🔶	bus2ip_burst	0	1										
- 🄶	busZip_burstlength	00) <u>oo</u>							DZ			00
- 🔶	busZip_wrreq	0	1										
-	bus2ip_rdreq	0	1		П		П						
-	bus2ip_cs	0)0)2		[[2	()2		12			0
-	bus2ip_rdce	0) 0		12		[[2	02		0			
- 🔶	busZip_wrce	0	<u>)</u> o							Z			0
- 🄶	ipZbus_data	0000000	<u>)000000000</u>				Xloot	1000(<u>(</u> 000)	100ľ	0000	0000		
- 🔶	ip2bus_wrack	0	1									Ц	Л
- 🔶	ip2bus_rdack	0	1										
- 🔶	ip2bus_addrack	0	1										.г
ø	ip2bus_error	0	1										

Ē

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



Creating Custom IP 14-25

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



Creating Custom IP 14-26

© Copyright 2013 Xilinx

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



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

IP Catalog X			🝌 Update IP Catalog 🛛 🛛 🗙
Search: Q- Name		∠1 Version ∇2	Update your IP Catalog with new user IP. Optionally specify the location of IP repositories.
Automotive & Industrial AXI Infrastructure AXI Infrastructure AXI Infrastructure AXI Infrastructure BaseIP Communication & Networking Debug & Verification Digital Signal Processing Debug & Verification Digital Signal Processing Debug & Processing Debug & Processing Debug & Storage Elements Memories & Storage Elements Digital Bus Interfaces Dideo & Image Processing	5	Customize IP	Xilinx IP Repository Search Path: C:/Xilinx_2012.3/14.3/ISE_DS/ISE Optional IP Repository Search Paths C:/My_IP_Repository C:/My_IP_Repository f Add Directories OK
		License Status Compatible Families Add IP Update IP Catalog Export to Spreadsheet	



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



XILINX > ALL PROGRAMMABLE.

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

1	🚴 IP Packager Summ	ary	×						
	The following information was found and added to your IP:								
	IP Identification								
	xilinx.com:user:syste	em_wrapper:1.0							
	IP Compatibility								
	Added compatibility f	or family: zyng							
	IP File Groups								
	Found 2 files for syn	thesis.							
	Top Module found fo	r synthesis: system_wrapper							
	Found 1 file for simul	ation.							
	Top Module found fo	r simulation: system_wrapper							
	IP Customization	Parameters							
	No parameters found	for IP customization.							
	IP Ports								
	Found 24 ports on y	our IP.							
	IP Interfaces								
	No bus interfaces fo	und for your IP.							
	IP GUI Customizat	tion Layout							
	Default IP GUI creat	ed.							
	Next Steps:	Walk through each of the items on the left that do not have a check mark or have an exclamation mark and verify the information populated.							
	IP Definition Location:	c:/xup/embedded/LED/component.xml							
		OK							

XILINX > ALL PROGRAMMABLE.

IP Packager

> Modify configuration

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

Σ Project Summary 🛛 🗙 🔞 axi_lite_sla	ve.v 🗙 🔮 Package IP	- axi_lite_slave ×	□ ٿ ×
✓ IP Identification	IP Identification		
V IP Compatibility	0 errors 0 wa Fill in and modif	nings 0 info messages y the information fields below that will be	used to identify your IP. Set the
💜 IP File Groups	categories in w	hich your IP will appear in the IP Catalog I	by modifying the Taxonomy.
IP Customization Parameters	Identification		
💜 IP Ports	Vendor :	xilinx.com	٢
💜 IP Interfaces	Library :	user	8
💜 IP Addressing and Memory	Name :	axi_lite_slave	0
💜 IP GUI Customization Layout	Version : 	1.0	
IP Licensing and Security	Display Name :	axi_lite_slave_v1_0	8
🎷 Review and Package	Description :	axi_lite_slave_v1_0	8
	Vendor Display Name :		
	Company Url :		
	Categories :	/Basic_Elements	-
	Root Directory :	c:/xup/embedded/labs/led_ip	
	Xml File Name :	c:/xup/embedded/labs/led_ip/component	xml
	Show advanced infor	mation	

IP repository

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

\lambda Project Settings	×
3	IP Repository Manager Packager
<u>G</u> eneral	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.
Simulation	IP Repositories
>	C:/xup/embedded/labs/led_ip (Project)
Synthesis	7
	Add Repository @ Refresh All
Bitstream	IP in Selected Repository
=	axi_lite_slave_v1_0 (xilinx.com:user:axi_lite_slave:1.0)
Īb	🔮 Add IP 🥏 Refresh Repository
	OK Cancel Apply

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

👃 Create And Package New IP		
	Welco Packa This wizar	ome to Create and age IP rd addresses:
	IP Packa Start the the sourc	aging: process of creating a piece of IP using e files and information from your current
	project d	🚴 Create And Package New IP 🛛 💽
2013.3	AXI4 Pe Creates includes example	Choose Create Peripheral or Package IP
	Click Nex	 Package your project Use the project as the source for creating a new IP Definition. Note: All sources to be packaged must be located at or below the specified directory. Package generated files - Package already generated HDL for IP in the project. Package a specified directory Choose a directory as the source for creating a new IP Definition.
		 Package as a library core - Library cores are available for reference by other IP. Note: Library cores do not appear in the IP Catalog.
		Create new AXI4 peripheral Creates AXI4 IP, Driver, TestApp and the AXI4 BFM example design.
		C:/xup/embedded/labs/lab2
		<pre>< Back Next > Finish Cancel</pre>

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 ind
     if ( S AXI WSTRB[byte index] == 1 ) begin
        // Respective byte enables are asserted as per write strobes
        // Slave register O
        slv req0[(byte index*8) +: 8] <= S AXI WDATA[(byte index*8) +: 8]</pre>
      end
 2'h1:
    for ( byte index = 0; byte index <= (C S AXI DATA WIDTH/8)-1; byte ind
     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]</pre>
      end
 2'h2:
    for ( byte index = 0; byte index <= (C S AXI DATA WIDTH/8)-1; byte ind
     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]</pre>
      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]</pre>
      end
```

XILINX > ALL PROGRAMMABLE.

HDL AXI Lite

Connect user logic to registers, or modify design



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}};</pre>
     //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}};</pre>
      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)
```

XILINX ➤ ALL PROGRAMMABLE.

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)

/*

* 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;
</pre>
```

XILINX > ALL PROGRAMMABLE.

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

