Creating and Adding Custom IP

Zynq
14.2 Version
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 peripherals using Create/Import IP wizard
Outline

- AXI4 Transactions
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - AXI4 Slave
  - AXI4 Master
- Create and Import Peripheral Wizard
- Incorporating your IP functionality
- 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**

- **Xilinx provides three configurable IPIC**
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - AXI4 Slave Burst

- **Xilinx AXI Reference Guide (UG761)**
## AXI4 Signals (AXI4, AXI4-Lite)

<table>
<thead>
<tr>
<th>AXI4</th>
<th>AXI4-Lite</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLK</td>
<td></td>
</tr>
<tr>
<td>ARESETN</td>
<td></td>
</tr>
<tr>
<td>AWID</td>
<td>AWADDR</td>
</tr>
<tr>
<td>AWLEN</td>
<td></td>
</tr>
<tr>
<td>AWSIZE</td>
<td></td>
</tr>
<tr>
<td>AWBURST</td>
<td></td>
</tr>
<tr>
<td>AWLOCK</td>
<td></td>
</tr>
<tr>
<td>AWCACHE</td>
<td></td>
</tr>
<tr>
<td>AWPROT</td>
<td></td>
</tr>
<tr>
<td>AWQOS</td>
<td></td>
</tr>
<tr>
<td>AWSIZE</td>
<td></td>
</tr>
<tr>
<td>AWREGION</td>
<td></td>
</tr>
<tr>
<td>AWLOCK</td>
<td></td>
</tr>
<tr>
<td>AWUSER</td>
<td></td>
</tr>
<tr>
<td>AWVALID</td>
<td></td>
</tr>
<tr>
<td>AWREADY</td>
<td></td>
</tr>
</tbody>
</table>

### Write Data

<table>
<thead>
<tr>
<th>AXI4</th>
<th>AXI4-Lite</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDATA</td>
<td>WDATA</td>
</tr>
<tr>
<td>WSTRB</td>
<td>WSTRB</td>
</tr>
<tr>
<td>WLAST</td>
<td>WUSER</td>
</tr>
<tr>
<td>WVALID</td>
<td>WREADY</td>
</tr>
<tr>
<td>BID</td>
<td>BRESP</td>
</tr>
<tr>
<td>BRESP</td>
<td>BUSER</td>
</tr>
<tr>
<td>BVALID</td>
<td>BREADY</td>
</tr>
</tbody>
</table>

### Read Address

<table>
<thead>
<tr>
<th>AXI4</th>
<th>AXI4-Lite</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARID</td>
<td>ARADDR</td>
</tr>
<tr>
<td>ARLEN</td>
<td>ARSIZE</td>
</tr>
<tr>
<td>ARBURST</td>
<td>ARLOCK</td>
</tr>
<tr>
<td>ARCACHE</td>
<td>ARQOS</td>
</tr>
<tr>
<td>ARPOT</td>
<td>ARREGION</td>
</tr>
<tr>
<td>ARUSER</td>
<td>ARVALID</td>
</tr>
<tr>
<td>ARREADY</td>
<td>RVALID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AXI4</th>
<th>AXI4-Lite</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>RDATA</td>
</tr>
<tr>
<td>RRESP</td>
<td>RRESP</td>
</tr>
<tr>
<td>RLAST</td>
<td>RUSER</td>
</tr>
<tr>
<td>RVALID</td>
<td>WREADY</td>
</tr>
</tbody>
</table>
Outline

- AXI4 Transactions
  - AXI4 Lite Slave
  - AXI4 Lite Master
  - AXI4 Slave
  - AXI4 Master
- Create and Import Peripheral Wizard
- Incorporating your IP functionality
- Summary
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
▶ Create and Import Peripheral Wizard
▶ Incorporating your IP functionality
▶ Summary
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
- Create and Import Peripheral Wizard
- Incorporating your IP functionality
- Summary
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

» Create and Import Peripheral Wizard

» Incorporating your IP functionality

» Summary
**AXI4 Master Burst – Block Diagram**

- **Parameterizable data width**
  - 32, 64, 128
- **Data burst**
  - 16, 32, 64, 128, 256 data beats
80 bytes of burst read
- ip2bus_mst_length signal

Transaction delimited by
- bus2ip_mstCmdack 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
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
Outline

➤ AXI4 Transactions
➤ Create and Import Peripheral Wizard
➤ Incorporating your IP functionality
➤ Summary
Create and Import Peripheral Wizard (CIP Wizard)

The wizard helps you create your own peripheral. It can also allow you to import a previously created peripheral into your design.

The wizard generates the necessary core description files into the user-selected directory.

You can start the wizard after creating a new project or opening an existing project in XPS.

The user peripheral can be imported directly through the wizard by skipping the creation option provided the peripheral already exists.

- Ensure that the peripheral complies with Xilinx implementation of AXI4 interface.
Starting the CIP Wizard

The Create and Import Peripheral Wizard can be started after creating a project and using Hardware → Create or Import Peripheral … or using Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.x → EDK → Tools → Create and Import Peripheral Wizard
Select the Flow and Directory

1. Select Create Peripheral flow

2. Select the target directory – project directory or user repository

The project directory assigned as the target directory will allow the peripheral to be available to the project without importing it. User repository will allow multiple projects to access the same peripheral by importing it in a project.
Selecting a Peripheral Name and Bus Interface

3. Provide the peripheral name and version

4. Select the interface to which the peripheral will attach
Selecting Various Services

5. Select services

6. Configure the SW accessible registers
Select IPIC Signals and BFM

7 Select the IPIC signals available to the user logic

8 Optional Bus Functional Model Simulation Template
You can select to
- generate HDL in Verilog
- generate project file so you can synthesize using XST and use ISE implementation tools
- software drivers
Since the project directory was assigned as the target directory the peripheral will appear in the IP Catalog under Project Local pcores folder.
Outline

- AXI4 Transactions
- Create and Import Peripheral Wizard
- Incorporating your IP functionality
- Summary
Entity statement
- Add custom port signals
- Add custom generics and parameters

Instantiate the rest of the design as a component

Review the sample code provided for each option
- Registers implemented for Bus2IP_WrCE and Bus2IP_RdCE selects
- Block RAM memory implemented for Bus2IP_CS
- Example code to generate interrupts
- Example code to transfer data between read/write FIFO

Modify/delete code to accommodate your application

Only the needed IPIC signals will appear in user_logic.vhd
IPIC Memory Read/Write

Read

Write

Bus2IP_Clk
Bus2IP_Addr
Bus2IP_BE
Bus2IP_CS(i)
Bus2IP_RNW
Bus2IP_Data
IP2Bus_Data
IP2Bus_RdAck
IP2Bus_WrAck
IP2Bus_Error

A0
D1
D2*
BE0
BE1
BE2*

D0
D1

A0
D1
D2
BE0
BE1
BE2
Outline

- AXI4 Transactions
- Create and Import Peripheral Wizard
- Incorporating your IP functionality
- 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 created and/or imported using Create/Import Peripheral wizard

The designer then needs to modify user_logic. vhd/v to include the desired functionality

Bring up all user proprietary signals up though the hierarchy to the top-level <ip_name>.vhd file

Modify mpd file to declare the user propriety signals so they are visible in System Assembly View

Modify pao file so lower-level modules are compiled