# Exploring AXI Transactions Using the AXI Traffic Generator

2023.2

## Abstract

Explore the AXI interface's channels, handshaking, and other signal members. The AXI Traffic Generator (ATG) IP example design and its simulation provide sample AXI traffic for study.

This lab should take approximately 45 minutes.

## CloudShare Users Only

You are provided with three attempts to finish a lab, where the time allotted to complete each lab is twice the expected completion time. Once the timer starts, you cannot pause the timer. Each lab attempt resets the previous attempt—your work from previous attempts is not saved.

## Objectives

After completing this lab, you will be able to:

* Generate an AXI Traffic Generator (ATG) core by using the IP catalog
* Simulate the ATG core example design
* Explain the purpose of the AXI4 channels and how read/write transactions with their AXI interface signals behave
* Program an ATG to generate simple AXI Lite transactions

## Introduction

First, you will study the AXI Traffic Generator (ATG) IP which can be found in the IP catalog in the Vivado™ Design Suite. The ATG IP component acts as an AXI protocol master, generating sample AXI traffic that can be used for both simulation and synthesis. You can implement this component in actual hardware and generate master-based AXI transactions. This lab will only demonstrate the simulation abilities of the core, but one could implement the design, download, and operate it in hardware.

This lab introduces you to the AXI4 memory interface and examines the simulation waveforms to observe the points like:

* All five AXI channels
* Key control interface signals
* Handshaking protocol
* Single and burst data beat read and write transactions

The ATG is a highly configurable block of logic with three basic modes of operation:

* AXI4 Traffic Generator: Allows for the creation of custom or pre-defined protocol (video, PCIe® interface, Ethernet, USB, or data) AXI full interface transactions.
* The custom sub-mode is built into the ATG and uses a block RAM that supports programming for up to 1024 reads and writes. This includes custom address, data, burst length, and other AXI signaling. This mode requires the user to program this internal ATG block RAM with the desired transaction pattern and ordering. The ATG's internal block RAM is loaded from another AXI master via an ATG slave AXI4-Lite port. After the block RAM is configured, the ATG can begin emitting AXI transactions.
* The example design in this lab uses this mode to generate the AXI4 transactions.
* AXI4-Lite Traffic Generator (System Init/Test Mode): Enables the creation of custom AXI-Lite transactions. The internals of this design contain up to four block RAM buffers that must be loaded via a bit file using \*.coe (coefficient) RAM initialization files. This ATG mode facilitates up to 256 AXI4-Lite read or write transactions. The example design uses a second ATG in this mode to program the full AXI4 ATG described above. This mode differs from the mode above in that:
* Only AXI4-Lite transactions can be generated (single data beat).
* No external programming of AXI transactions is necessary (this is performed via COE block RAM initialization files) as opposed to the above AXI4 Traffic Generator mode that requires the ATG internal block RAM buffers be programmed via an ATG slave AXI4-Lite port (which is suppressed in this mode).
* Less flexibility in transaction generation.
* AXI4-Stream Traffic Generator: Generates and receives AXI streaming interface traffic. This lab does not use this mode.

The ATG is a very complex and flexible IP and its full explanation is beyond the scope of this lab. Documentation for the ATG is covered in the AXI Traffic Generator Product Guide (PG125), which is available from the Documentation Navigator tool.

The intention of this lab is to illustrate the use of the Vivado Design Suite tools to generate the AXI Traffic Generator base example design and demonstrate use of the Vivado simulator. Although conceptually simple, it is left to the student to perform a detailed examination of the design RTL and simulation testbench outside of this lab.

The following is the simplified block design for the simulated design. These components are described below.

A diagram of a computer

Description automatically generated

Figure 3-1: AXI Traffic Generator Example Design Block Diagram

The HDL example design contains these items:

* Driver: An instantiation of the ATG in AXI4-Lite mode. This module is used to generate AXI4 Lite transactions to program the DUT module.
* DUT: Device under test. A second instantiation of an ATG, this time in AXI4 mode. This module is used to generate AXI4 transaction to the Responder.
* Responder: An instantiation of a block RAM controller that will accept the generated traffic from the ATG. This differs from the block RAM transaction buffers that are internal to the two ATGs.

The design uses two instances of the ATG, one in AXI4-Lite mode (Driver) and the other in AXI4 mode (DUT). Both are instantiated as IP using the XCI file format. When a component's XCI source is opened, the component Re-customize IP dialog box launches. Beginning with the device under test (DUT), you will see how it is customized here:

A screenshot of a computer

Description automatically generated

Figure 3-2: DUT – ATG AXI4 Re-customize IP Dialog Box

The DUT is configured to generate AXI4 traffic based on contents of its internal RAM. Read and write transactions are both emitted from the master interface that connects to the Responder as shown in the block diagram. The ATG in AXI4 custom mode contains internal block RAMs for storing traffic content and control registers, both of which are accessed over an AXI4-Lite slave interface. After the ATG's control registers and RAM transaction buffers are set up, traffic generation begins when the core\_ext\_start input port is active.

When the ATG is in AXI4 Traffic Generator mode, internal RAM is divided into four sections:

* Command RAM (CMDRAM): Two regions of up to 256 AXI transactions commands each, one for read and one for writes. See the description below.
* Parameter RAM (PARAMRAM): Two regions of up to 256 parameters to modify CMDRAM-generated transactions. Not used in this lab.
* Master RAM (MSTRAM): Write and read data buffer. See description below.
* Address RAM (ADDRRAM): Upper address bits for traffic addresses greater than 32 bits. Not used in this lab.

Each of these block RAM buffers must be filled via the AXI4-Lite slave interface by a processor or some other mechanism. When the core\_ext\_start input port is activated, a sequencer, internal to the ATG, starts at the beginning of each of the four above buffers, stepping through each entry to generate AXI transactions on the master AXI port of the ATG. The slave interface address map is shown below.

A table with numbers and letters

Description automatically generated

Figure 3-3: Slave Interface Address Map

The CMDRAM is where all the work takes place. It is a buffer that contains control information to generate custom AXI transactions. The buffer is organized into two regions of 256 entries of 128 bits, one read and one write for AXI4 transaction generation. The 128-bit control word is the same for each.

For this lab, only the bits of interest will be described. Access to the CMDRAM is over the AXI4-Lite slave interface with 32-bit data transactions to the slave base address + region offset (see the above two figures).

Below is the address map of the CMDRAM buffer showing the read and write regions. When the ATG is enabled, it cycles from the beginning to the end of the buffer, generating an AXI4 transaction based on the 128-bit command and its 32-bit modifier in PARAMRAM. Both read and write regions generate transactions simultaneously (an AXI port can have simultaneous traffic over its read and write channels).

A white sheet of paper with black text

Description automatically generated

Figure 3-4: CMDRAM Address Map

The encoding of the 128-bit command entry is shown below in 32-bit word chunks, as to be written and accessed over the AXI4-Lite slave interface. Those entries of interest for this lab are highlighted. You may wish to return to this table for reference.

A screenshot of a computer

Description automatically generated

Figure 3-5: CMDRAM Memory Format

The first 32-bit word is used as the low order 32 bits of address for the AXI-generated transaction (1). Bit 31 of the second word is used to enable the ATG to begin generating traffic (2), and bits [7:0] set the transaction burst length (3).

The other ATG RAM buffer region of interest in the DUT is the master RAM (MSTRAM), used for AXI read and write channel generation. Data is taken from this buffer for write transactions and stored during read transactions. As mentioned earlier, the CMDRAM and MSTRAM buffers must be filled via the ATG AXI4-Lite slave interface by a processor or some other mechanism. Since this design does not have a processor involved, another mechanism option will be used.

The Driver block is attached to the AXI4-Lite slave interface of the DUT and programs the CMDRAM and other block RAM buffers in the DUT. One of the easiest ways to configure an ATG in a design without a processor is with another ATG. The Driver component is nothing more than the ATG in AXI4-Lite mode, as shown below in its Re-Customize IP dialog box.

A screenshot of a computer

Description automatically generated

Figure 3-6: Driver – ATG AXI4 Re-customize IP Dialog Box

In System Test mode the ATG's four internal block RAM buffers are pre-programmed during GUI core configuration via four COE files. While these buffers differ in format from the previously discussed AXI4 mode, the ATG similarly sequences through them to generate transactions on the AXI4-Lite master interface. This mode can be used in a system without a processor to initialize the system peripherals with preconfigured values on system reset or for simple system testing.

After the core comes out of reset in System Test mode, it reads the coefficient (COE) files (address, data, control, and mask) from the ROM and generates AXI4-Lite transactions. You must provide the COE files for this mode. Entries in the COE files are 32 bits.

A description of the COE files is as follows:

* Address COE file: Provides the sequence of addresses to be issued.
* Data COE file: Provides the sequence of data corresponding to the address specified in the address COE file. Data in this file is written for AXI4-Lite transactions and compared to incoming data during reads.
* Control COE file: Provides various transaction fetch control, error checking enabling, and read/write commands. For the context of this lab, you will only be concerned if the bit signifies a read or write.
* Mask COE file: Used only for read transactions to indicate which bits of returning read data should be checked against the Data COE file.

Each entry in the corresponding COE file generates a single AXI4-Lite transaction. The number of entries in all COE files must be the same.

Authoring these four COE files is like writing a program to talk to an AXI slave. The allowed number of file entries is 16, 32, 64, 128, and 256. You can insert NOP (No Operation) defined by address (0xFFFFFFFF) in the middle of a COE address file. The core stops generating further transactions (including the current NOP address of 0xFFFFFFFF) after the NOP is encountered.

You need to ensure that at least one NOP address is present in the address COE file. Typically, one would pad the address COE with NOPs after the last useful address to meet the number of file entries requirement.

Operation:

1. After the AXI Traffic Generator comes out of reset, it reads the first entry in all four buffers.
2. It initiates AXI4-Lite write transactions to a specified address, data, control, and mask entries in the COE files, which are now in the buffers.
3. It sequences through all four buffers, generating AXI4-Lite transactions, until a NOP address is encountered.
4. The core goes to an idle state after AXI4-Lite transactions are issued.

Your design is configured so that the COE files program the DUT AXI4 ATG to generate AXI4 traffic to the Responder - block RAM controller and its associated block RAM, as shown below in its Re-Customize IP dialog box.

A screenshot of a computer

Description automatically generated

Figure 3-7: Responder - AXI4 Block RAM Re-customize IP Dialog Box

The example design provided here is a structural design consisting of two ATGs and a block RAM controller—no other RTL is needed.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux® platform.

One environment variable is required: TRAINING\_PATH, which points to the location of the lab files. This variable comes configured in the CloudShare/CustEd\_VM environments.

Some tools can use this environment variable directly (that is, $TRAINING\_PATH is recognized and automatically expanded), and some tools require manual expansion (/home/amd/training for the CloudShare/CustEd\_VM environments). The lab instructions describe what to do for each tool. Other environments require the definition of this variable for the scripts to work properly.

The Vivado™ Design Suite and the Vitis™ Unified IDE offer a Tcl environment used in many labs. When either tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from any previous launch of the tools.

If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

Nomenclature

Formal nomenclature is used to explain how different arguments are used. The following are some of the more commonly used symbols:

| Symbol | Description | Example | Explanation |
| --- | --- | --- | --- |
| <text> | Indicates a field | cd <dir> | <dir> represents the name of the directory. The < and > symbols are NOT entered. If the directory to change to is XYZ, then you would enter cd XYZ into the environment. |
| [text] | Indicates an optional argument | ls [ | more] | This could be interpreted as ls <Enter> or ls | more <Enter>. The first instance lists the files in the current Linux directory, and the second lists the files in the current Linux directory, but additionally runs the output through the more tool, which paginates the output. Here, the pipe symbol (|) is a Linux operator. |
| | | Indicates choices | cmd <ZCU104 | VCK190> | The cmd command takes a single argument, which could be ZCU104 OR VCK190. You would enter either cmd ZCU104 or cmd VCK190. |

## General Flow

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step 1:  Creating the Placeholder Project | A grey arrow pointing to the right  Description automatically generated | Step 2:  Generating  the Core  and Design | A grey arrow pointing to the right  Description automatically generated | Step 3:  Simulating & Analyzing Transactions |

Creating the Vivado Design Suite Placeholder Project Step

Many IP blocks provide their own reference design. As you will be working with the example design for the AXI Traffic Generator, you will create a "placeholder" project that does nothing more than enable you to access the specific piece of IP and launch another instance of the Vivado Design Suite containing the example ATG design.

The ZCU104 (Zynq™ UltraScale+™ MPSoC), VCK190 (Versal™ adaptive SoC), or ZC702 (Zynq 7000 SoC) boards are supported in this lab as hardware design targets; however, any board (or part) could be used, as the FPGA fabric is common to all.

Admittedly, there are differences in the FPGA fabric itself, but at the level that this lab focuses on, this is inconsequential. There is no synthesis or implementation of the design in this lab (even though the design is capable of synthesis and implementation)—only simulation.

Begin the process by creating a placeholder project. This project acts as a gateway to the IP example designs.

Here are two ways to open the Vivado Design Suite.

1-1. Open the Vivado Design Suite.

1-1-1. Click the Vivado icon () from the taskbar.

A red arrow pointing to a black box

Description automatically generated

Figure 3-8: Launching the Vivado Design Suite from the Taskbar

Note: It takes a few moments to launch. The order of the icons in your environment may be different.

Alternatively, open the Linux terminal window (<Ctrl + Alt + T>) and enter the following:

source /opt/amd/Vivado/2023.2/settings64.sh; vivado

Note: This installation path is valid for the CustEd VM and CloudShare environments. Use the proper path for your environment.

The tool opens with a Welcome window. From here you can create a new project, open an existing project, enter Tcl commands, and access documentation and examples.

1-1-2. [Optional] Maximize the window as there is a lot of information to see.

A screenshot of a computer

Description automatically generated

Figure 3-9: Vivado Design Suite Welcome Screen

Hint: If the Tcl Console is not visible, double-click the Tcl tab to make it visible.

"Create Project" is the starting point for all designs. Projects contain sources, settings, graphics, IP, and other elements that are used to build a final bitstream and analyze a design. The Create New Project Wizard in the Vivado Design Suite allows you to specify HDL and other project resource files that will be included in the project.

1-2. Create a new, blank Vivado Design Suite project.

1-2-1. Click Create Project to begin the process (1).

A screenshot of a computer

Description automatically generated

Figure 3-10: Creating a New Vivado Design Suite Project

This will launch the New Project Wizard.

1-2-2. Click Next to exit the introductory dialog box and begin entering project-specific information (2).

1-3. Describe the various aspects of the project.

1-3-1. Enter ip\_placeholder in the Project name field (1).

1-3-2. Enter the following location in the Project location field (2):

$TRAINING\_PATH/AXItransactions/lab

Important: The Vivado Design Suite is capable of expanding variables when running under both the Linux and Windows environments. The available environment variables (regardless of the OS) must be predicated with the '$' symbol.

Alternatively, you can use the browse feature to navigate to where you want the project to reside.

1-3-3. Uncheck the Create Project Subdirectory option if it is selected (3).

Leaving this checked will create an unnecessary level of hierarchy for the lab.

A screenshot of a computer

Description automatically generated

Figure 3-11: Entering the Project Name and Location

1-3-4. Click Next to advance to the next dialog box (4).

Here you will specify your project type as either an RTL project or a post-synthesis project. An RTL project enables you to add or create new HDL files and synthesize them, whereas the post-synthesis project requires pre-synthesized files. When an empty design is created, an RTL project is used.

1-3-5. Select RTL Project (1).

1-3-6. Select Do not specify sources at this time to instruct the Vivado tool to create a blank project (2).

While existing sources could be entered at this time, you will enter them later so that you can move through this portion of the project creation process more quickly.

A screenshot of a computer

Description automatically generated

Figure 3-12: Specifying Project Options

1-3-7. Click Next to advance to the target device/platform selection (3).

1-4. Select the target part by first filtering by the board name. If you are not targeting a supported board, you will need to filter by the part.

1-4-1. Click Boards from the Default part area to filter by the board type rather than by the specific part (1).

1-4-2. Select xilinx.com from the Vendor drop-down list in the Filter area (2).

This limits the number of boards seen to those manufactured by the specified vendor.

1-4-3. Select ZCU104 (MPSoC), VCK190 (Versal), or ZC702 (Zynq 7000) from the board list.

If you accidentally click the hyperlink, a web page will open for that board. You can close the browser page.

If the board you want to use is not immediately visible, click the Refresh button to update the board catalog.

Note: While the web page contains important information and resources for the board, these details are not needed to complete this lab.

A screenshot of a computer

Description automatically generated

Figure 3-13: Selecting the Board for the Project

1-4-4. Click Next to advance to the summary (3).

A summary of your project is displayed. If you want to change any of the information that you entered, you can do so now by clicking Back until you reach the correct dialog box. After the project is created, the project properties can still be edited.

1-4-5. Click Finish to accept these settings and build the project.

Your project is constructed and leaves you in the operational portion of the Vivado Design Suite GUI.

1-5. Verify that the Vivado Design Suite project language is set to Verilog, which is the default for this example design.

Selecting VHDL, while valid, would generate another hierarchical wrapper layer to the underlying Verilog RTL.

1-5-1. Select Tools > Settings to open the Project Settings dialog box.

A screenshot of a computer

Description automatically generated

Figure 3-14: Selecting Project Settings

Note: This same capability is accessible from the Flow Navigator > Project Manager > Settings.

1-5-2. If necessary, select Verilog from the Target language drop-down list to select it as the base HDL language used for template creation (1).

A screenshot of a computer

Description automatically generated

Figure 3-15: Selecting the Target Language

1-5-3. Click OK (2).

Question

When creating the Vivado Design Suite project, why was a board chosen rather than a part?

Generating the ATG Core and Example Design Step

The ATG IP core needs to be added to the empty project so that the example design can be generated. The IP catalog wizard will be used to create and parameterize the AXI Traffic Generator core and add it to the placeholder project.

While you are generating the core, the default parameters will be accepted since the values do not matter (the core just needs to exist in the project). Once the ATG core has been added, the example design for this core can be generated. At that point a second Vivado Design Suite project, containing the ATG example design, will be created and opened.

2-1. Access and configure the AXI Traffic Generator via the IP catalog.

2-1-1. Under the Flow Navigator, select Project Manager > IP Catalog to open the catalog.

A screenshot of a computer

Description automatically generated

Figure 3-16: Opening the IP Catalog

2-1-2. Expand the Embedded Processing > Debug & Verification > Debug folders to locate the IP core of interest.

2-1-3. Double-click AXI Traffic Generator to both select the IP and open the Customize IP dialog box.

A screenshot of a computer

Description automatically generated

Figure 3-17: Selecting the AXI Traffic Generator IP Core

If you encounter a dialog box asking you to "Add the IP to the Block Design", "Customize IP", or "Cancel", then select Customize IP.

Note: This usually occurs for Windows users.

The Customize IP Wizard opens for the AXI Traffic Generator.

A screenshot of a computer

Description automatically generated

Figure 3-18: ATG Customization Dialog Box

2-1-4. Click OK to add the ATG IP to the design using the default settings.

The Generate Output Products dialog box appears.

A screenshot of a computer

Description automatically generated

Figure 3-19: Skipping Output Product Generation

2-1-5. Click Skip because output products are not needed at this time.

Question

In what form is the ATG IP added to the ip\_placeholder Vivado Design Suite project?

2-2. Generate the example design for the AXI Traffic Generator.

2-2-1. From the Sources > Hierarchy pane, select the axi\_traffic\_gen\_0 component under Design Sources to begin the process of generating the example design for this IP.

2-2-2. Right-click axi\_traffic\_gen\_0 to open the context menu.

2-2-3. Select Open IP Example Design.

A screenshot of a computer

Description automatically generated

Figure 3-20: Generating the Example Design

The Open IP Example Design dialog box opens.

The Example project directory field should be populated with the current project director.

2-2-4. If it is not, either manually enter $TRAINING\_PATH/AXItransactions/lab, or browse to this directory by using the  icon (1).

This is the folder where the example design will be placed.

A screenshot of a computer

Description automatically generated

Figure 3-21: Opening the IP Example Design

2-2-5. Click OK to begin generating the design (2).

2-2-6. If you are asked for permission to create the example project directory, click OK.

The example design takes several minutes to build. The completion of the build will be indicated when a new Vivado Design Suite project opens containing the example design and the status in the upper-right corner shows "Ready".

A computer screen with a red box and text

Description automatically generated

Figure 3-22: Vitis IDE Showing "Ready" Status

Question

How many Vivado IDE projects are now open? How do they differ?

2-3. Return to the ip\_placeholder project Vivado tool instance.

The purpose of the ip\_placeholder project was to access the IP. Now that the example design is open in a new Vivado IDE session, you will return to the placeholder project so that you can close it.

2-3-1. Select the ip\_placeholder project.

Be careful not to accidentally close the ATG example design project.

A screenshot of a computer

Description automatically generated

Figure 3-23: Identifying the ip\_placeholder Project

Hint: The project name is shown in the title bar.

2-4. Close the Vivado Design Suite.

2-4-1. Select File > Exit.

The Exit Vivado dialog box opens.

A screenshot of a computer

Description automatically generated

Figure 3-24: Exit Vivado Dialog Box

2-4-2. If you are asked to save the project or a portion of the project, select whichever elements of the project you want to save, then click Save to save the selected elements; otherwise, click Don't Save.

2-4-3. Click OK when you are asked to exit the Vivado Design Suite.

Note: You can choose to select the Don't show this dialog again option to avoid being asked for confirmation when exiting the Vivado Design Suite.

2-5. Study the structure of the ATG example design that was opened as a Vivado Design Suite project during the example design creation process. View the source hierarchy of the example design.

The project opens with the Design tab selected. You will now investigate the portion of the design in the PL by switching to the Sources tab.

2-5-1. Return to the axi\_traffic\_gen\_0\_ex Vivado tool instance.

2-5-2. Click the Expand All icon () in the Sources > Hierarchy pane to expand the source tree.

Note the synthesizable RTL and simulation testbench sources as shown in the figure below.

A screenshot of a computer

Description automatically generated

Figure 3-25: Example Design Source Hierarchy

Question

Why does the synthesizable design source RTL show up in two places in the hierarchy?

2-5-3. Double-click the design top-level RTL component, axi\_traffic\_gen\_0\_exdes, to open it.

2-5-4. Examine the RTL structure.

Note: The Re-Customize IP dialog box for each of the IP catalog XCI components can be opened by double-clicking them.

Question

List the three major components of the top-level RTL design. (Reference the block diagram at the beginning of this lab along with the source hierarchy.)

Question

Which of the three components are really ATG cores?

2-6. Open a COE file to examine its contents.

Note that the data format will be in binary format, which is difficult to read. You will replace these files with the ones provided in hexadecimal format. This will make them easier to modify later in this lab.

2-6-1. From the Sources window, expand the Coefficient Files group if necessary.

A screenshot of a computer

Description automatically generated

Figure 3-26: Expanding the Coefficient Files Group

2-6-2. Double-click addr.coe to open the default binary-formatted file in the editor.

Note the structure of the file, including the variable names memory\_initiializaiton\_radix and memory\_initialization\_vector. This binary format does not make understanding the meaning of the data any easier.

A screenshot of a computer

Description automatically generated

Figure 3-27: Binary COE File

2-6-3. Click the X next to the file name on the tab to close the file.

2-7. Remove the binary COE files from the project.

COE files can also be written in hexadecimal, which is much easier to read. New COE files have been prepared for you to replace the binary versions so that it is easier for you to modify.

2-7-1. Using the Sources pane, right-click addr.coe to open the context menu.

2-7-2. Select Remove File from Project to delete this COE file from the project.

A screenshot of a computer

Description automatically generated

Figure 3-28: Deleting the COE Files

2-7-3. When the Remove Sources dialog box opens, click OK to confirm deletion.

A screenshot of a computer error

Description automatically generated

Figure 3-29: Confirming Deletion

2-7-4. Select ctrl.coe and shift-click mask.coe to select the remaining COE files (ctrl.coe, data.coe, and mask.coe).

2-7-5. Press the <Delete> key to delete these files.

2-7-6. Click OK to confirm the deletion.

2-8. Add the new hexadecimal-formatted COE files.

2-8-1. From the Sources pane, expand the axi\_traffic\_gen\_0\_exdes source if it is not already expanded (1).

2-8-2. Right-click the driver component to open the context menu (2).

Note: This refers to the component. The full name is driver: atg\_lite\_agent.

2-8-3. Select Re-customize IP to open the IP configuration dialog box (3).

2-8-4. From the COE File Paths group and adjacent to the description of each COE file, use the browse button to navigate to $TRAINING\_PATH/AXItransactions/support (4, 5).

2-8-5. Replace each of the existing paths with the new hexadecimal COE file of the same name (4).

That is, addr.coe is used for the Address COE File path, data.coe is used for the Data COE File path, etc.

Hint: You will need to climb the hierarchy several levels to find the support directory (5).

A screenshot of a computer

Description automatically generated

Figure 3-30: Replacing the COE Files for the Driver ATG

2-8-6. Click OK to accept the values and close the dialog box (6).

The Generate Output Products dialog box opens.

2-8-7. Click Skip because output products are not needed at this time.

A screenshot of a computer

Description automatically generated

Figure 3-31: Skipping Output Product Generation

The following instructions will have you modify some of the COE files. These hexadecimal-formatted COE files have been modified from the original to enhance the lab experience.

A modified simulation waveform file has been provided. The modified waveform file includes added signals to view that were not in the original.

HDL simulation files can be added to the design at any time.

2-9. Add simulation files to the design.

2-9-1. Select Add Sources under Project Manager in the Flow Navigator.

A screenshot of a computer

Description automatically generated

Figure 3-32: Selecting Add Sources

2-9-2. Select Add or create simulation sources.

A screenshot of a computer

Description automatically generated

Figure 3-33: Add Sources Dialog Box

2-9-3. Click Next.

2-9-4. Click the Plus () icon to open the pop-up menu.

2-9-5. Select Add Files to open the Add Source Files dialog box which allows you to browse to the desired directory.

2-9-6. Browse to the $TRAINING\_PATH/AXItransactions/support directory if it is not open already.

2-9-7. Select axi\_traffic\_gen\_0\_tb\_top\_behav.wcfg.

2-9-8. Click OK in the Add Source Files dialog box.

2-9-9. Ensure that the Copy sources into project option is selected.

2-9-10. Click Finish to add the file(s) to the project.

Simulating the Design and Analyzing the AXI Transactions Step

Now that the example design is complete, it can be simulated and AXI transactions studied. The testbench provides clock, reset, and Driver start signaling stimuli.

A brief overview and block diagram of the design can be found in the Introduction section of this lab. The Driver is an ATG that generates AXI4-Lite traffic based on the contents of the four COE files. The Driver AXI4-Lite master port attaches to the DUT ATG slave AXI port and is used to configure its block RAM-based controller/traffic generator logic. After the DUT has been configured, the last write to its control register (bit 20 ='1') enables traffic generation.

The base design will be simulated first and then you will modify the Driver ATG COE files to modify the AXI traffic generation in the ATG DUT.

3-1. When the COE files are updated, it is necessary to reset the project simulation output products. Otherwise, the simulation will run with the old COE files. Since the COE files are only used for the Driver ATG component only that component needs to be reset.

Note: Every IP catalog-generated component has its own set of output products.

3-1-1. Using the Sources pane, right-click the driver: atg\_lite\_agent component to open the context menu.

3-1-2. Select Reset Output Products.

A screenshot of a computer

Description automatically generated

Figure 3-34: Reset Driver ATG Component Output Products

The Reset Output Products dialog box appears.

3-1-3. Click Reset to reset the output products.

A screenshot of a computer

Description automatically generated

Figure 3-35: Confirming Output Products Reset

3-2. Run behavioral simulation using the Vivado simulator.

3-2-1. Select Simulation > Run Simulation > Run Behavioral Simulation under Simulation from the Flow Navigator.

The simulation window opens, and the simulation runs for the length of time specified in the Simulation Settings (1 us default).

You can alternatively start the Vivado simulator by entering launch\_simulation in the Tcl command line.

3-2-2. After the waveform opens, set the run duration to 15 us in the single cell in the top toolbar (1).

3-2-3. Click the Run for icon to execute the simulation for 15 us (2).

The simulation terminates at the end of the test. The testbench file opens, showing the last line of code executed.



Figure 3-36: Setting the Simulation Run to 15 us

3-2-4. Select the axi\_traffic\_gen\_0\_tb\_top\_behav.wcfg tab to return to the waveform view.

Note that double-clicking the tab will enlarge the view to full screen so that you can more easily navigate the design.

3-2-5. Click the Zoom All icon () to see the entire simulation.

3-2-6. Optional: Use the various controls to zoom and position the waveform cursor. You can "float" the waveform window (), maximize to full screen (), or adjust the column width for maximum waveform exposure.

Note: If you are unfamiliar with the Vivado simulator, hovering the mouse pointer over an icon in the Waveform toolbar describes its functionality.

A screenshot of a computer

Description automatically generated

Figure 3-37: ATG Design Simulation Waveform

The testbench writes operational activities and messages to the Tcl Console (1). These aid with understanding how the design is behaving, including providing a summary of the test results (2).

A screenshot of a computer

Description automatically generated

Figure 3-38: Simulation Console Messages

3-3. Use the various zoom and pan controls to view the waveform and answer the questions below.

3-3-1. Examine the various names of the waveform signal group dividers.

Question

What do the waveform signal group dividers represent?

Question

How many AXI ports are indicated? Which components are their masters?

Question

Examine the signal names under each channel and their related waveform activity. What two signal names are common to each channel? How do they seem to operate?

3-3-2. Use the waveform controls to zoom into the region from 1.5 us to 1.7 us.

3-3-3. Examine the first two Driver AXI4-Lite transactions.

Question

How is the beginning of a transaction identified? What is the first transaction?

Question

What is the second transaction?

Note that there are many Driver write transactions that are configuring the DUT to generate AXI traffic to the Responder. The DUT does not start generating transactions until it has been configured and its Master Control register (0x00000000) start command (bit 20 equal '1') has been written to.

3-3-4. Use the waveform controls to zoom to 13.535 us to examine this transaction.

A screenshot of a computer

Description automatically generated

Figure 3-39: Enabling DUT – ATG to Begin Traffic Generation

Note that the rising edge valid signal indicates that a channel's information is good.

3-3-5. Use the waveform controls to pan to 14 us.

You will see the Driver ATG polling the DUT ATG until Driver indicates that all the traffic has been generated. Note the flexibility of the ATG in supporting these different modes.

A screenshot of a computer program

Description automatically generated

Figure 3-40: DUT – ATG Indicating Completion of Traffic Generation

You will now turn your attention to the AXI4 full traffic generated by the DUT to the Responder.

3-3-6. Pan to starting at 13.645 us, the address of the first DUT transaction, and examine the remaining DUT transactions.

Question

How many AXI4 full transactions are generated by the DUT ATG? What type are they?

These transactions make up the entire DUT traffic program provided with the example design and are based on how the Driver configured the DUT from the information in the COE files.

Notice that after the traffic generation is completed the Driver, which monitors a competition bit in the DUT Error Status register, asserts a Done and Status signal. The testbench program monitors these signals and ends the simulation with the appropriate console message.

Question

What seems to be different about the Driver transactions compared to the DUT transactions? (Hint: Look at the data channels.)

The DUT write (or read) address channel has a signal, m\_axi\_awlen[7:0] (arlwen for reads), that indicates the number of data beats that will occur on the data channel for the transaction.

3-3-7. Using the wready, wvalid, rready, and rvalid signals, determine how many data beats there are for each of the five DUT-based transactions.

Question

What values of awlen and arlen are driven, as burst length, for each DUT transaction? How many data beats are there in each transaction's data channel? What is the relationship between the number of data beats and the length? Fill the values in the table below.

| Transaction | Type | Address | Number  Data Beats | Burst Length  awlen or arlen |
| --- | --- | --- | --- | --- |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |

3-3-8. Examine the second write and read transactions (transactions two and five).

Note that the Responder component is block RAM (memory).

Question

What address is being written to and read from? Is the same data being read that was written?

3-4. Unload or exit the simulation but do not exit from the Vivado Design Suite itself.

3-4-1. Select File > Close Simulation to exit from the simulator.

3-4-2. Click OK to confirm if needed.

3-5. Change the AXI traffic that the DUT generates.

This is performed by altering the COE files that are used by the Driver component to configure the DUT. You will begin by examining the contents of the address COE file to see some of the configuration addressing. The data COE file will then be opened, and you will change the DUT ATG configuration settings regarding burst length.

3-5-1. From the Sources window, expand the Coefficient Files folder.

3-5-2. Double-click addr.coe to open it.

The address COE file contains a list of addresses that the Driver ATG reads and writes. A read or write operation coding is determined by the corresponding location in the ctrl.coe file (beyond the scope of this lab). Likewise write data (or predicted read results) is contained in the corresponding location in the data.coe file. You will not be modifying anything in the address COE file.

Currently, as shown below, the DUT is being configured to generate the five AXI4 transactions. Each transaction requires 128 bits, written by four dword writes by the Driver. There are two identical data structures in the DUT ATG. The one beginning at 0x00008000 is for read and the one at 0x00009000 is for writes. There are two read entries and three write entries.

The coding of each entry is beyond the scope of this lab. Additional information for this can be found through DocNav in the Traffic Generator Product Guide (PG125).

A screenshot of a computer

Description automatically generated

Figure 3-41: Driver Address COE File

3-5-3. Close the addr.coe tab to avoid accidentally corrupting this file.

3-5-4. Double-click data.coe to open it.

Hint: If this file is not visible, expand the Coefficient Files folder again.

Each entry in the data COE file corresponds to the address entry in the address COE file.

A screenshot of a computer

Description automatically generated

Figure 3-42: Unmodified Driver Data COE File

Question

Line three in the address and data COE files represent the first AXI4-Lite transaction that was emitted by the Driver. What does it represent? (Hint: It is encoded as a read.)

3-5-5. Change the lines in the data.coe file as follows:

| Line # | Value | Explanation |
| --- | --- | --- |
| 4 | 00000040 | Address of first read transaction |
| 8 | 00000080 | Address of second read transaction |
| 9 | 80002404 | Burst length (m\_axi\_arlen) of second read transaction |
| 13 | 80002403 | Burst length (m\_axi\_awlen) of first write transaction |
| 17 | 80002401 | Burst length (m\_axi\_awlen) of second write transaction |
| 21 | 80002404 | Burst length (m\_axi\_awlen) of third write transaction |

Note: These values are in hexadecimal.

A screenshot of a computer

Description automatically generated

Figure 3-43: Modified Driver Data COE File

3-5-6. Select File > Text Editor > Save File to save the changes to data.coe.

3-6. Reset the output products and rerun the simulation.

3-6-1. These processes are outlined in instructions 3-1 and 3-2.

3-7. Study the resulting traffic from the DUT ATG.

3-7-1. Examine the DUT write and read addresses.

Question

Fill in the values of the table below, comparing the results to those in the previous table that you completed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transaction | Type | Address | Number  Data Beats | Burst Length  awlen or arlen |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |

3-7-2. Zoom in to around 13 us.

3-7-3. Observe the second DUT write.

Note the DUT write transaction to address 0x00000040 with a burst awlen of 0x01.

3-7-4. Study the write data channel data, ready, and valid signals.

A screenshot of a computer

Description automatically generated

Figure 3-44: Second DUT – ATG Write

Question

How many data beats are there on the write data channel? How can you tell? What is the significance of the ATG dropping ready for a clock cycle?

Since the Responder (driver by the DUT) is a block RAM memory controller, one would expect data written to a memory location would be the same when read back from that location.

3-7-5. Examine the DUT address and data channels, paying attention to the actual address and data values.

Question

Examine the DUT write and read transactions. Is the read data consistent with what was written to those transaction addresses?

Note that the burst length of writes and reads to the same addresses are different.

Question

Do different burst lengths of the reads and writes affect the data value outcomes?

3-8. Exit from the Vivado Design Suite.

3-8-1. Select File > Exit to close the Vivado Design Suite.

3-8-2. Click OK.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the $TRAINING\_PATH/AXItransactions directory.

3-9. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

3-9-1. Navigate to $TRAINING\_PATH/AXItransactions.

3-9-2. Select AXItransactions.

3-9-3. Press <Delete>.

-- OR --

[Linux users]: Using the command line:

3-9-4. Press <Ctrl + Alt + T> to open a terminal window.

3-9-5. Enter the following command to delete the contents of the workspace:

[host]$ rm -rf $TRAINING\_PATH/AXItransactions

## Summary

You used the Vivado Design Suite IP catalog to generate an example design for the AXI Traffic Generator core. The design was simulated and AXI transactions were studied. You then changed the generated AXI traffic by modifying the contents of the COE files that drive the ATG core. With this limited exposure to the ATG core features, you are now in a position to further explore use of this core in your own applications.

## Answers

1. When creating the Vivado Design Suite project, why was a board chosen rather than a part?

The ZCU104 was chosen (as opposed to an FPGA part) to take advantage of the IP catalog core creation features to generate constraints for this hardware platform when the example design is to be opened. For a custom board, you would have specified the FPGA family, part, package, and speed grade instead.

1. In what form is the ATG IP added to the ip\_placeholder Vivado Design Suite project?

The core is represented in the Design Sources folder (in the Sources pane) as an XCI file. This was generated by the IP catalog wizard and represents the core. This file can be instantiated as a component in an RTL source. Doubling-clicking the file will open the Re-customize IP dialog box, allowing the core parameters to be modified and updated.

1. How many Vivado IDE projects are now open? How do they differ?

There are two Vivado IDE projects now open. The originally created project (ip\_placeholder.xpr) is just a dummy project so that the IP catalog could be launched and the AXI Traffic Generator core generated. Subsequently, this project is not used. The second project (axi\_traffic\_gen\_0\_example.xpr) is the example design project that was opened in the last step.

1. Why does the synthesizable design source RTL show up in two places in the hierarchy?

The RTL, axi\_traffic\_gen\_0\_exdes, shows under the Design Sources folder for synthesis and in the Simulation Sources folder for simulation. It is the same RTL reference in both locations.

1. List the three major components of the top-level RTL design. (Reference the block diagram at the beginning of this lab along with the source hierarchy.)

Verilog modules:

* atg\_lite\_agent driver: Instantiation of the ATG in AXI4-Lite (System Init) mode.
* axi\_traffic\_gen\_0 DUT: Instantiation of the ATG in AXI4 mode.
* bram\_memory responder: Block RAM target for AXI-generated transactions.

1. Which of the three components are really ATG cores?

The Driver and the DUT components are both ATG cores. The Driver is configured to be an AXI4-Lite ATG and the DUT is configured as an AXI4 full ATG.

1. What do the waveform signal group dividers represent?

They represent the five various AXI channels. Each divider contains a few of the more important signals associated with that channel. The waveform would appear as too busy if all of the signals were shown.

1. How many AXI ports are indicated? Which components are their masters?

As suggested by the divider names, there are two AXI ports. The Driver ATG component generates AXI4-Lite transactions (the first five channels). The DUT-labeled channels represent the AXI4 full port driving the Responder.

1. Examine the signal names under each channel and their related waveform activity. What two signal names are common to each channel? How do they seem to operate?

Each channel has a ready and valid signal. These are the main handshaking signals across the AXI connection used to transfer the information (address, data, response, and/or control) across the channel.

Valid indicates that information is present and ready indicates information acceptance. Information is transferred on the rising edge of s\_axi\_clk when both are a '1'. The AXI channel side generates the valid signal while the receiver generates the ready signal.

1. How is the beginning of a transaction identified? What is the first transaction?

The beginning of any AXI transaction is activity on the write or read address channel. The first transaction is a read from address 0x00000000 on the read address channel followed by a return of 0x20000000 on the read data channel. This transaction is reading the Master Control register (location 0x00000000) of the DUT ATG. The 0x20 (high byte) represents the revision of the ATG, which is defined in the AXI Traffic Generator Product Guide (PG125).

1. What is the second transaction?

The second transaction is a write to 0x00008000, a value of 0x00000000, which will be the first address of the AXI traffic that will be generated by the DUT. Hence the DUT is being configured for the AXI traffic that will be generated to the Responder.

1. How many AXI4 full transactions are generated by the DUT ATG? What type are they?

A total of five transactions are generated: three write followed by two read.

1. What seems to be different about the Driver transactions compared to the DUT transactions? (Hint: Look at the data channels.)

The Driver port only provides for a single data transfer for either a read or write operation. This is because it is only AXI4-Lite capable. The DUT port is AXI4 full and is capable for multiple data beats (transfers) per transaction.

1. What values of awlen and arlen are driven, as burst length, for each DUT transaction? How many data beats are there in each transaction's data channel? What is the relationship between the number of data beats and the length? Fill the values in the table below.

| Transaction | Type | Address | Number  Data Beats | Burst Length  awlen or arlen |
| --- | --- | --- | --- | --- |
| 1 | write | 0x00000000 | 3 | 2 |
| 2 | write | 0x00000040 | 4 | 3 |
| 3 | write | 0x00000080 | 4 | 3 |
| 4 | read | 0x00000000 | 3 | 2 |
| 5 | read | 0x00000040 | 4 | 3 |

The AMBA AXI and ACE Protocol Specification (AMBA AXI and ACE Protocol.pdf located in the support directory) defines the burst length as the value of awlen (arlen) - 1. Reference page A3-44.

1. What address is being written to and read from? Is the same data being read that was written?

Both transactions are writing and reading address 0x00000040. The data burst length is 3 and the data is identical for the write and read.

1. Line three in the address and data COE files represent the first AXI4-Lite transaction that was emitted by the Driver. What does it represent? (Hint: It is encoded as a read.)

As you previously studied, this first transaction emitted by the Driver is a read of address 0x00000000, the DUT ATG Master Control register. The read returns a 0x20000000, the version of the ATG core.

1. Fill in the values of the table below, comparing the results to those in the previous table that you completed.

| Transaction | Type | Address | Number  Data Beats | Burst Length  awlen or arlen |
| --- | --- | --- | --- | --- |
| 1 | write | 0x00000000 | 4 | 3 |
| 2 | write | 0x00000040 | 2 | 1 |
| 3 | write | 0x00000080 | 5 | 4 |
| 4 | read | 0x00000040 | 3 | 2 |
| 5 | read | 0x00000080 | 5 | 4 |

1. How many data beats are there on the write data channel? How can you tell? What is the significance of the ATG dropping ready for a clock cycle?

The burst length, m\_axi\_awlen, is 0x01, meaning that there are two data beats on the channel. This is verified by noting that m\_axi\_wready and m\_axi\_wvalid are '1' for two clock cycles.

This is a good example of ready/valid handshaking. In this example, the AXI block RAM controller for some reason needed an extra clock cycle before receiving the second data and pulled a wait state by dropping the ready signal for that clock cycle.

1. Examine the DUT write and read transactions. Is the read data consistent with what was written to those transaction addresses?

Yes.

1. Do different burst lengths of the reads and writes affect the data value outcomes?

No, data value outcomes depend on the starting address and not the burst length. It is the task of the AXI agent to store the channel address and increment it for each data beat in the burst. Whether it is a write or a read does not matter.