

Exploring AXI Transactions Using the AXI Traffic Generator

2016.3

Abstract

AXI4 transactions will be explored in this lab with special emphasis on AXI channels, handshaking, and the most useful signal members within the AXI interface. The AXI Traffic Generator (ATG) IP example design will serve as the basis of this lab. Simulation of the design will provide the sample AXI traffic to be studied.

Objectives

After completing this lab, you will be able to:

- Generate an AXI Traffic Generator (ATG) core by using the IP catalog
- Simulate the Xilinx-provided (ATG) core example design
- Explain the purpose of the AXI4 channels and how read/write transactions with their AXI interface signals behave

Introduction

The main focus of this lab is to introduce you to the AXI4 (memory/full) interface, including:

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

You will examine simulation waveforms to observe the above points. A design is needed to simulate to accomplish this end. The AXI Traffic Generator (ATG) IP (which can be found in the IP catalog) has been selected for this lab. The ATG component documentation includes a Xilinx-provided sample design that will be the basis of this lab.

The ATG IP component acts as a 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.

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

- AXI4 Traffic Generator:** This mode allows for the creation of custom or protocol (choice of video, PCIe® interface, Ethernet, USB, or data) AXI (full interface) transactions. The custom sub-mode uses block RAM supporting programming for up to 1024 reads and writes. This includes custom address, data, burst length, and other AXI signaling. It is necessary to program the ATG block RAM with the desired transaction pattern and ordering via a slave AXI4-Lite port. The example design in this lab uses this mode to generate the AXI4 transactions that will be studied.
- AXI4-Lite Traffic Generator (System Init/Test Mode):** This mode allows for the creation of custom AXI (Lite interface) 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 designs 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 is necessary (this is performed via COE block RAM initialization files)
 - Less flexibility in transaction generation
- AXI4-Stream Traffic Generator:** This mode 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 included in the *support* directory for your reference.

The intention of this lab is to illustrate the use of the Vivado® Design Suite tools to generate the Xilinx-provided 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.

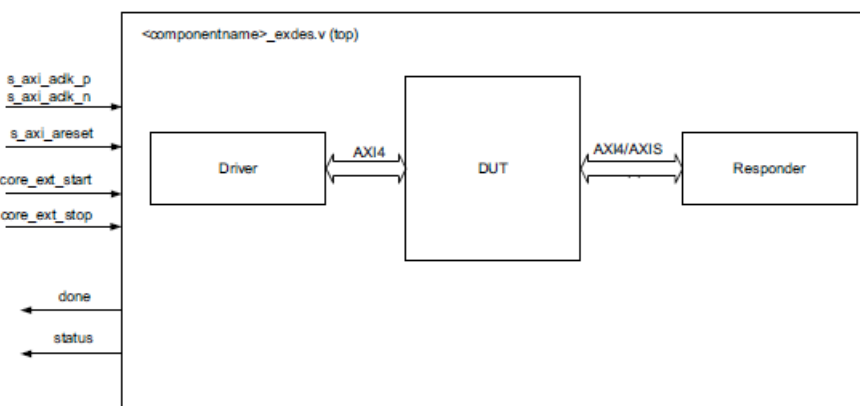


Figure 3-1: AXI Traffic Generator Example Design Block Diagram

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. An instantiation of the ATG 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.

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 either 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:

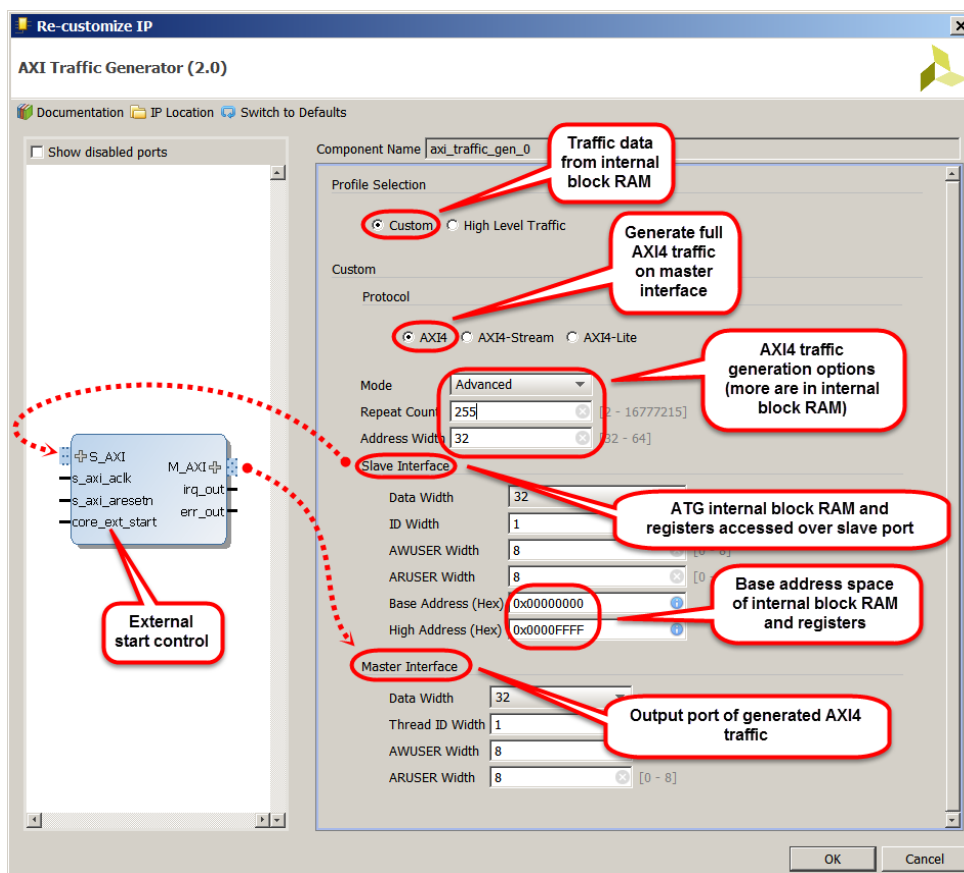


Figure 3-2: DUT – ATG AXI4 Re-customize IP Dialog Box

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 is connected 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. The control registers are configured to generate traffic when the core_ext_start port is active.

The 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. The slave interface address map is shown below.

Region	Description
0x0000_0000–0x0000_0FFF	Internal registers
0x0000_1000–0x0000_17FF	PARAMRAM (2 KB)
0x0000_8000–0x0000_9FFF	CMDRAM (8 KB)
0x0000_A000–0x0000_AFFF	ADDRRAM (2 KB)
0x0000_C000–0x0000_DFFF	MSTRAM (8 KB)

Figure 3-3: Slave Interface Address Map

The CMDRAM is where all the work takes place. It is a buffer that contains 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 the purpose of 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 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 command and its 32-bit modifier in PARAMRAM. Both read and write regions generate transactions simultaneously.

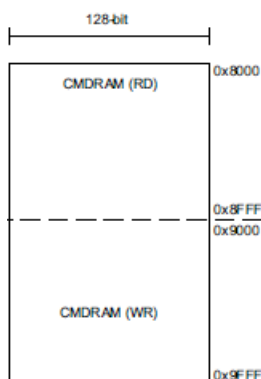


Figure 3-4: CMDRAM Address Map

The encoding of the 128 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. During the course of this lab, you may wish to return to this table for reference.

Word Offset	Bits	Description
+00	31:0	AXI Address[31:0]: Address to drive on ar_addr or aw_addr (a*_addr[31:0]) 1
	31	Valid_cmd ^[4] : When set, this is a valid command. When clear, halt the master logic for this request type (read or write).
	30:28	last_addr[2:0]: Should be set to 0 for C_M_AXI_DATA_WIDTH > 64 writes, indicates the valid bytes in the last data cycle. 2 64-bit mode: 000 = All bytes valid 001 = Only Byte 0 is valid 010 = Only Bytes 0 and 1 are valid ... 32-bit mode: 000 = All bytes valid 100 = Only Byte 0 is valid 101 = Only Bytes 0 and 1 are valid 110 = Only Bytes 0, 1, and 2 are valid
+01	27:24	Reserved
	23:21	Prot[2:0]: Driven to a*_prot[2:0]
	20:15	Id[5:0]: Driven to a*_id[5:0]
	14:12	Size[2:0]: Driven to a*_size[2:0]
	11:10	Burst[1:0]: Driven to a*_burst[1:0]
	9	Reserved
	8	Lock: Driven to a*_lock
	7:0	Len[7:0]: Driven to a*_len[7:0]. 3
+02	31	Reserved
	30:22	My_depend[8:0]: This command does not begin until this master logic has at least completed up to this command number. A value of zero in this field means do not wait. This allows a command to wait until previous commands have completed for ordering.
	21:13	Other_depend[8:0]: This command does not begin until the other master logic has completed up to this command number. For example, if a write command had 0x04 in this field, the write would not begin until the read logic had at least completed its commands (CMDs) 0x00 through 0x03. A value of 0 in this field means do not wait, but commands can only be started in order for each master type. For example, if Write CMD[0x05] waits for Read 0x03, then Write CMD[0x06] cannot start until Read 0x03 completes as well. A read completes when it receives the last cycle of data, and a write completes when it receives BRESP.
	12:0	Mstram_index[12:0] ^[2] : Index into MSTRAM for this transaction (reads will write to this MSTRAM address, writes take data from this address)
+03	31:20	Reserved
	19:16	qos[3:0]: Driven to a*_qos[3:0]
	15:8	user[7:0]: Driven to a*_user[7:0]
	7:4	cache[3:0]: Driven to a*_cache[3:0]
	3	Reserved
	2:0	Expected_resp: 0x0 to 0x1 = Only OKAY is allowed 0x2 = Only EX_OK is allowed 0x3 = EX_OK or OKAY is allowed 0x4 = Only DECERR or SLVERR is allowed 0x7 = Any response is allowed

Figure 3-5: CMDRAM Memory Format

The first 32-bit word is used as the low order 32 bits of 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 block RAM buffer of interest in the DUT is the master RAM (MSTRAM). Data is taken from this RAM 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, the *some other mechanism* option will be used.

In the block diagram, the Driver is the block that is attached to the AXI4-Lite slave interface of the DUT and programs the CMDRAM and other block RAM buffers in the DUT. As it turns out, 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 show below in its Re-Customize IP dialog box.

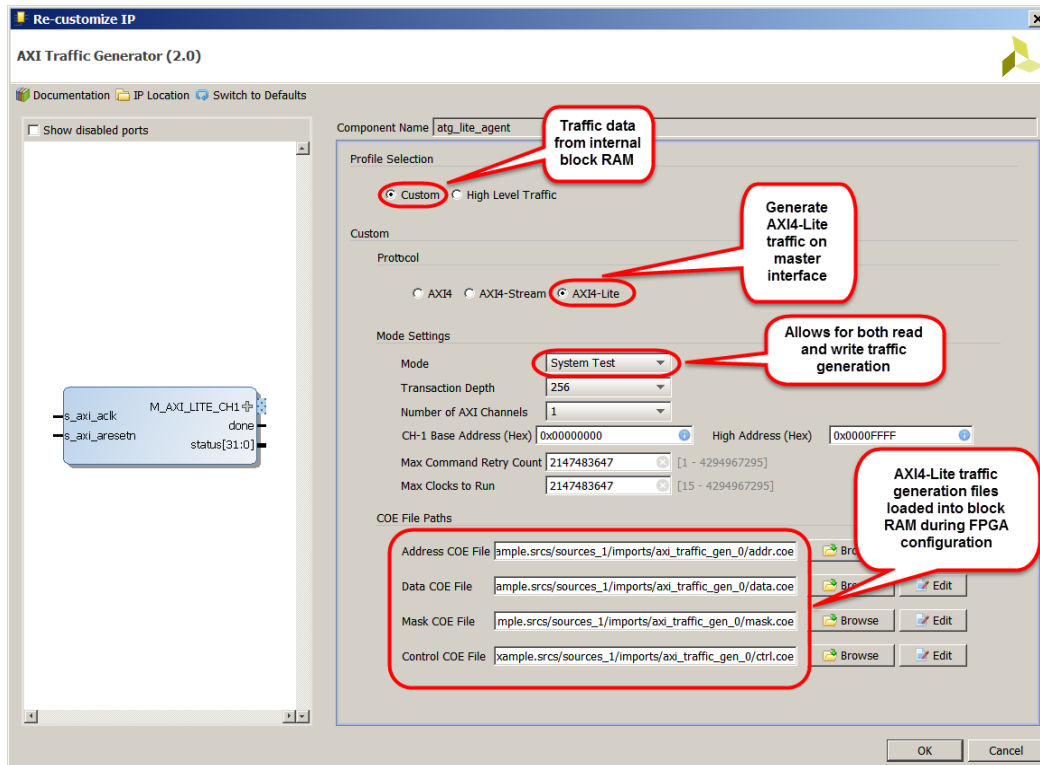


Figure 3-6: Driver – ATG AXI4 Re-customize IP Dialog Box

System Init mode is a special mode where the core provides an 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 Init mode, it reads the coefficient (COE) files (address, data, control, and mask) from the ROM and generates AXI4-Lite transactions. You must provide this COE files for this mode. Entries in all of 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.
- 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 ready 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. The number of entries in the COE file are user programmable. Allowed values are 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 address is present. You need to ensure that at least one NOP address is present in the address COE file.

Operation:

1. After AXI Traffic Generator comes out of reset, it reads the ADDR and DATA ROMs.
2. It initiates AXI4-Lite write transactions to a specified address and data in the COE files.
3. The core goes to an idle state after AXI4-Lite transactions are issued.

Your design is configured so that the COE files are set up to program the DUT AXI4 ATG to enable it to generate AXI4 traffic to the Responder, which is a block RAM controller (including the associated block RAM), as shown below in its Re-Customize IP dialog box.

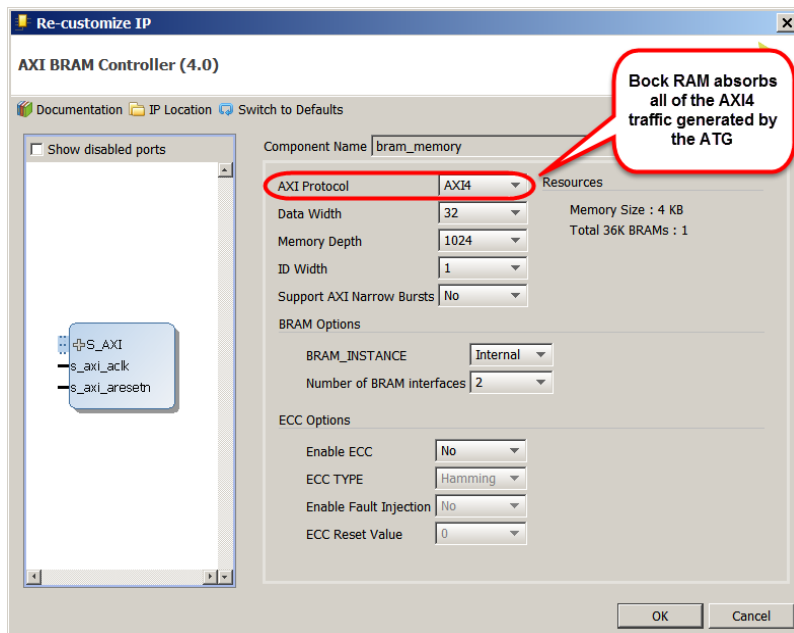
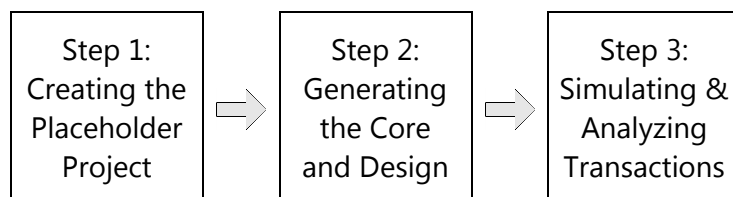


Figure 3-7: Responder - AXI4 Block RAM Re-customize IP Dialog Box

To summarize, the example design is a structural design consisting of two ATGs and a block RAM controller—no other RTL is needed.

General Flow



Creating the Vivado Design Suite Placeholder Project

Step 1

Many pieces of IP have their own reference design. Since 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 version of the Vivado Design Suite with the example ATG design.

While the ZC702 or ZedBoard board will be selected for this lab as a hardware design target, any board (or part) could be used, as the FPGA fabric is common to all. There is no synthesis or implementation of this design (even though the design is capable of synthesis and implementation) in this lab, only simulation.

There are a number of ways to launch the Vivado Design Suite. The two most popular mechanisms are shown here.

1-1. Launch the Vivado Design Suite.

This can be done in two standard ways, use your preferred method.

1-1-1. Select **Start > All Programs > Xilinx Design Tools > Vivado 2016.3 > Vivado 2016.3.**

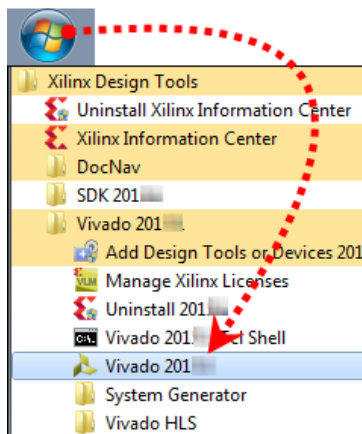


Figure 3-8: Launching the Vivado Design Suite from the Start Menu

-- OR --

Double-click the **Vivado Design Suite** shortcut icon () on the desktop.

The Vivado Design Suite opens to the Welcome window. From the Welcome window you can create a new project, open an existing project, or enter Tcl commands directly into the Vivado Design Suite as well as access documentation and examples.

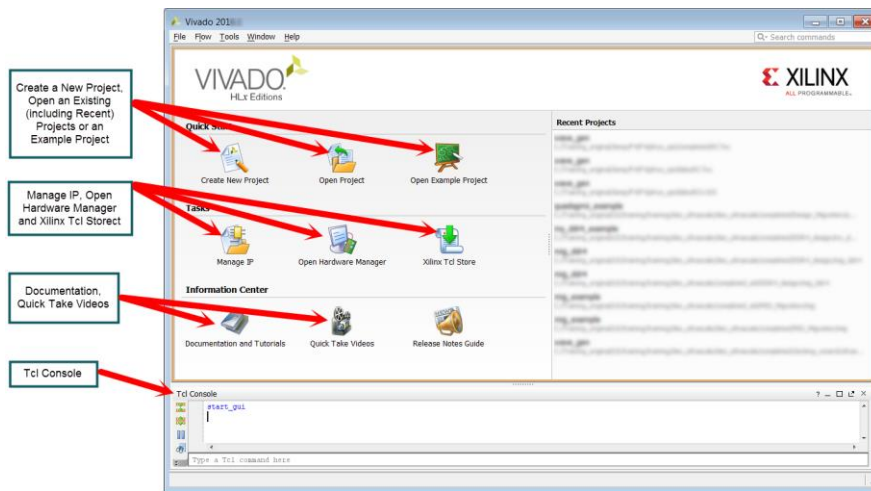


Figure 3-9: Vivado Design Suite Welcome Screen

"Create New Project" is the starting point for all designs. Projects contains 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 New Project** to begin the process (1).

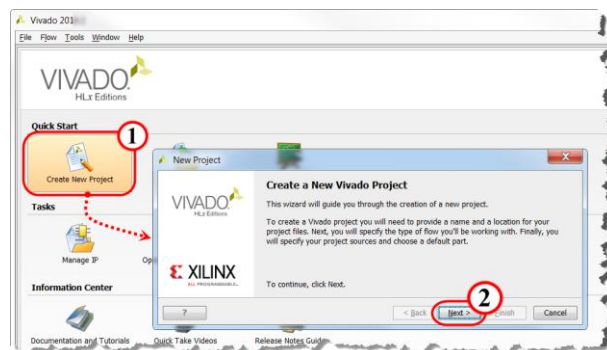


Figure 3-10: Creating a New Vivado Design Suite Project

This will launch the New Project Wizard.

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

1-3. You will now encounter a series of dialog boxes asking you to enter different pieces of information describing the project.

1-3-1. Enter **ip_placeholder** in the Project name field.

1-3-2. Enter **C:\training\AXItransactions\lab** in the Project location field.

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

1-3-3. Deselect the **Create Project Subdirectory** option as leaving this checked will create an unnecessary level of hierarchy for this lab.

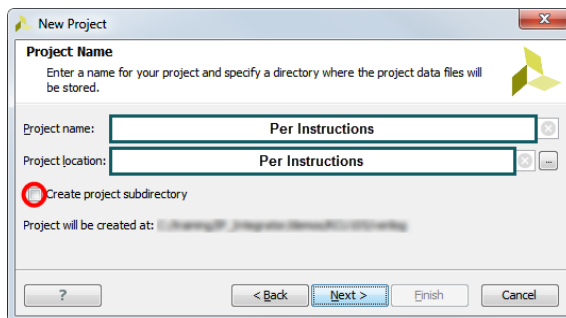


Figure 3-11: Entering the Project Name and Location

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

Here you will choose between an RTL project or a post-synthesis project. Simply put, 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** (2), which creates a blank project.

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.

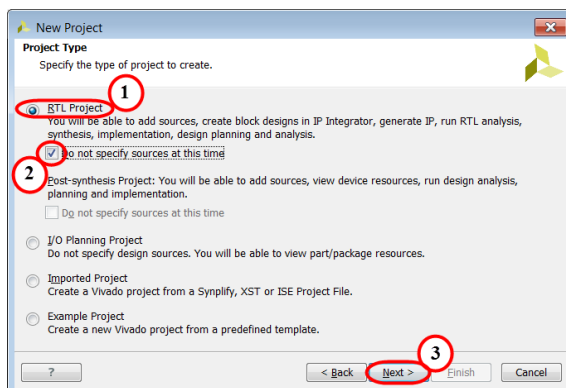


Figure 3-12: Selecting Project Type

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

1-4. Select the target part by first filtering by board and then by family. If you are not using a supported board, you will need to filter by part.

1-4-1. Select **Boards** from the Select area to filter by board rather than by the specific part (1).

1-4-2. Select **All** 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 **ZC702 or ZedBoard** from the board list.

Alternatively you can select the board directly from the list at any time while in this dialog box.

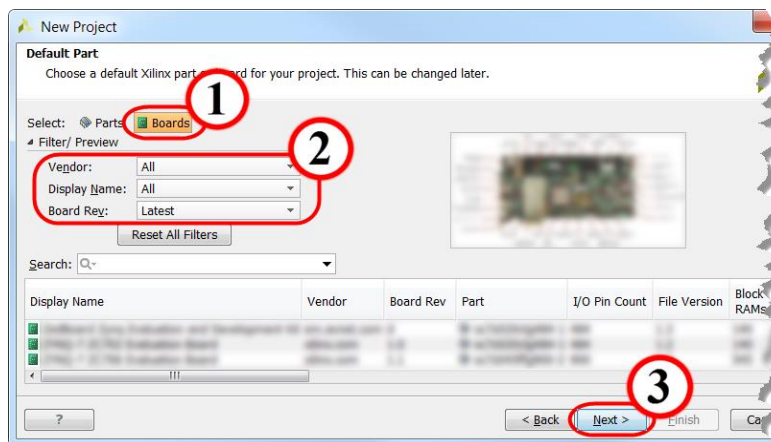


Figure 3-13: Selecting the Board for the Project

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 and making the correction, or you can create the project now and edit the project properties, add or remove files, etc. later.

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 default, Verilog, is selected for this lab example. Verilog is the native language for the example design. Selecting VHDL, while valid, would generate another hierarchical wrapper layer to the underlying Verilog RTL.

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

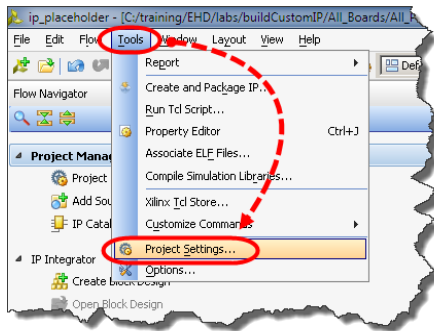


Figure 3-14: Selecting Project Settings

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.

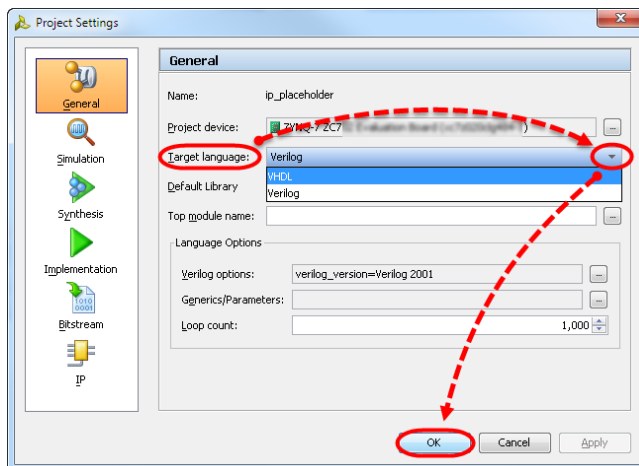


Figure 3-15: Selecting VHDL

1-5-3. Click **OK** to accept the changes.

Question 1

Why was a board chosen (as opposed to an FPGA part), when the Vivado Design Suite project was created?

Generating the ATG Core and Example Design

Step 2

The ATG IP core needs to be added to the empty project in order to generate the Xilinx-provided example design. 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 Xilinx 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. Open the IP catalog and add the AXI Ethernet Subsystem core to the empty Vivado Design Suite project.

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

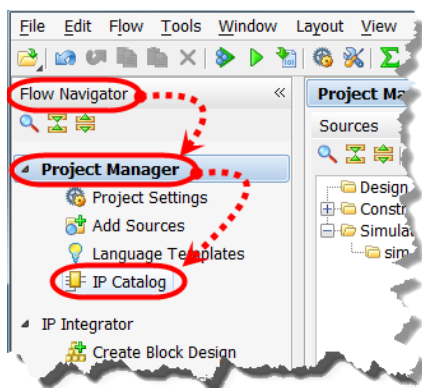


Figure 3-16: Opening the IP Catalog

- 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 its re-customization IP dialog box.

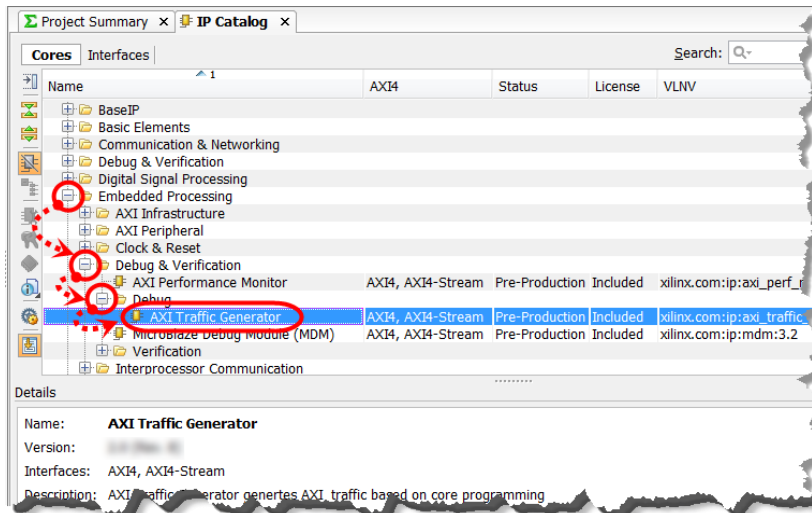


Figure 3-17: Selecting the AXI Traffic Generator IP Core

The Customize IP wizard opens for the AXI Traffic Generator.

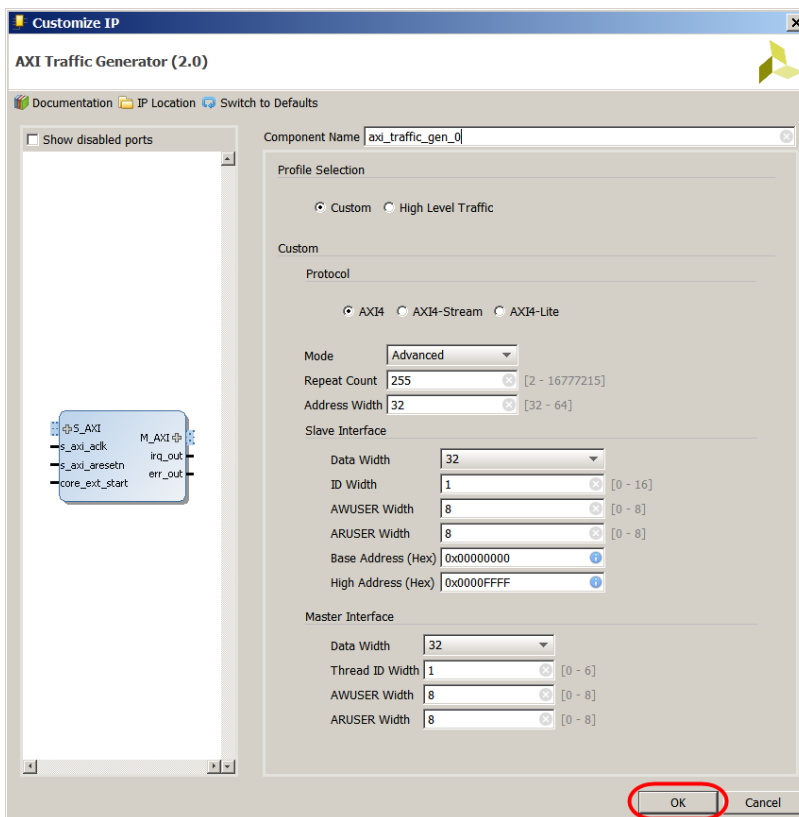


Figure 3-18: ATG Customization Dialog Box

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

The Generate Output Products dialog appears.

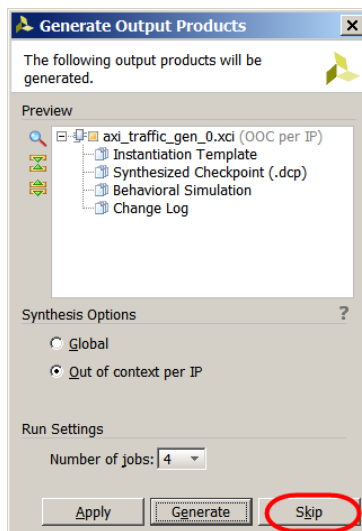


Figure 3-19: Generate Output Products Dialog Box

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

Question 2

In what form is the ATG IP added to the *ip_placeholder* Vivado Design Suite project?

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

2-2-1. From the Sources > Hierarchy pane, select the **axi_traffic_gen_0** component to begin the process to generate the Xilinx-provided Vivado IDE 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**.

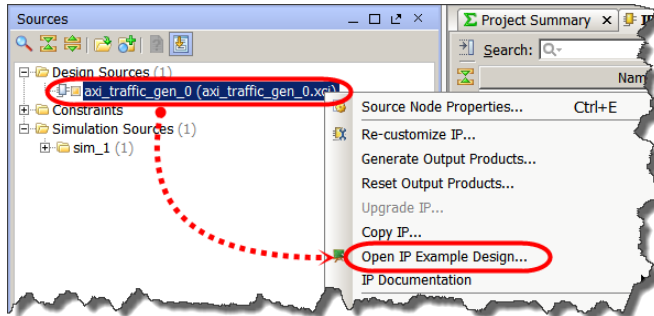


Figure 3-20: Generating the Example Design

The Open IP Example Design dialog box opens.

2-2-4. Click the  icon to select the **C:/training/AXItransactions/lab** directory (1).

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

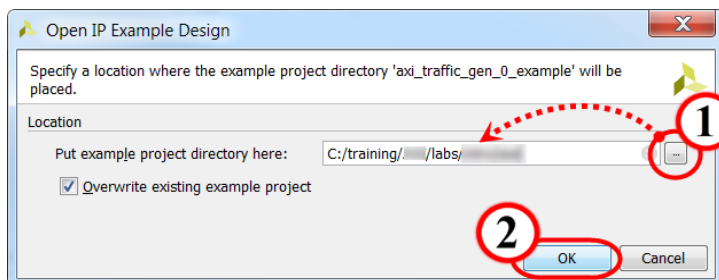


Figure 3-21: Open IP Example Design

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

2-2-6. Click **OK** again to accept creation of the example project directory.

The example design may take a couple of minutes to build. The completion of the build will be indicated when a new Vivado Design project opens containing the example design.

Question 3

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

2-3. The Vivado Design Suite *ip_placeholder* project is not required anymore.

2-3-1. Select the *ip_placeholder* project.

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

2-4. Close the project.

2-4-1. Select **File > Close Project** to close the project.

The Close Project dialog box opens.

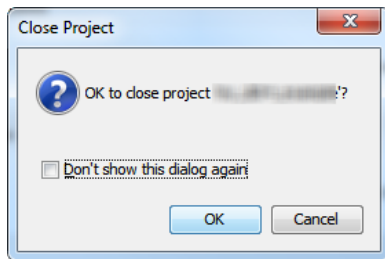


Figure 3-22: Close Project Dialog Box

2-4-2. Click **OK**.

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.

2-5-1. Click the **Expand All** icon (📁) in the Sources > Hierarchy pane to expand the source tree.

2-5-2. Identify the location of the top-level RTL and simulation testbench sources.

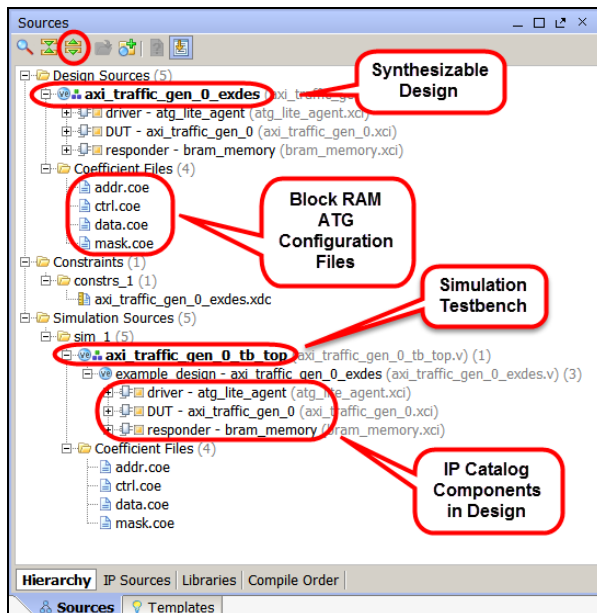


Figure 3-23: Example Design Source Hierarchy

Question 4

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 5

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 6

Which of the three components are really just 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** folder.

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

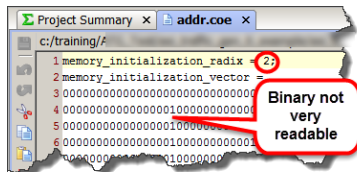


Figure 3-24: Binary COE File

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

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.

2-6-4. Using the Sources pane, right-click **addr.coe** and select **Remove File from Project** to delete it.

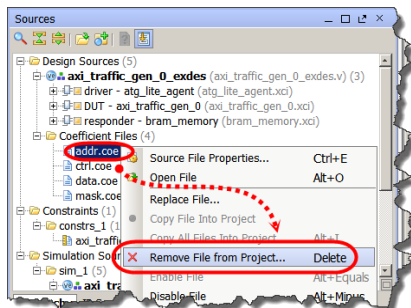


Figure 3-25: Deleting the COE Files

2-6-5. When the Remove Sources dialog box opens, click **OK** to confirm deletion.

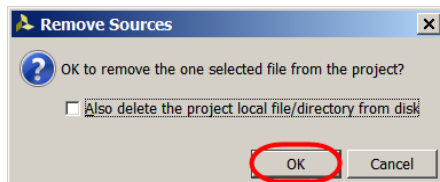


Figure 3-26: Confirming Deletion

2-6-6. Multi-select **ctrl.coe**, **data.coe**, and **mask.coe**.

2-6-7. Press the **<Delete>** key to delete these files.

2-6-8. Click **OK** to confirm the deletion.

2-7. Add the new hexadecimal formatted COE files.

- 2-7-1. From the Sources pane, right-click the **driver** component to open the context menu.
- 2-7-2. Select **Re-customize IP** to open the IP configuration dialog box.
- 2-7-3. For each of the four files that were deleted, browse to `C:\training\AXItransactions\support` and replace the existing COE files.
- 2-7-4. Click **OK** to accept these new COE files.

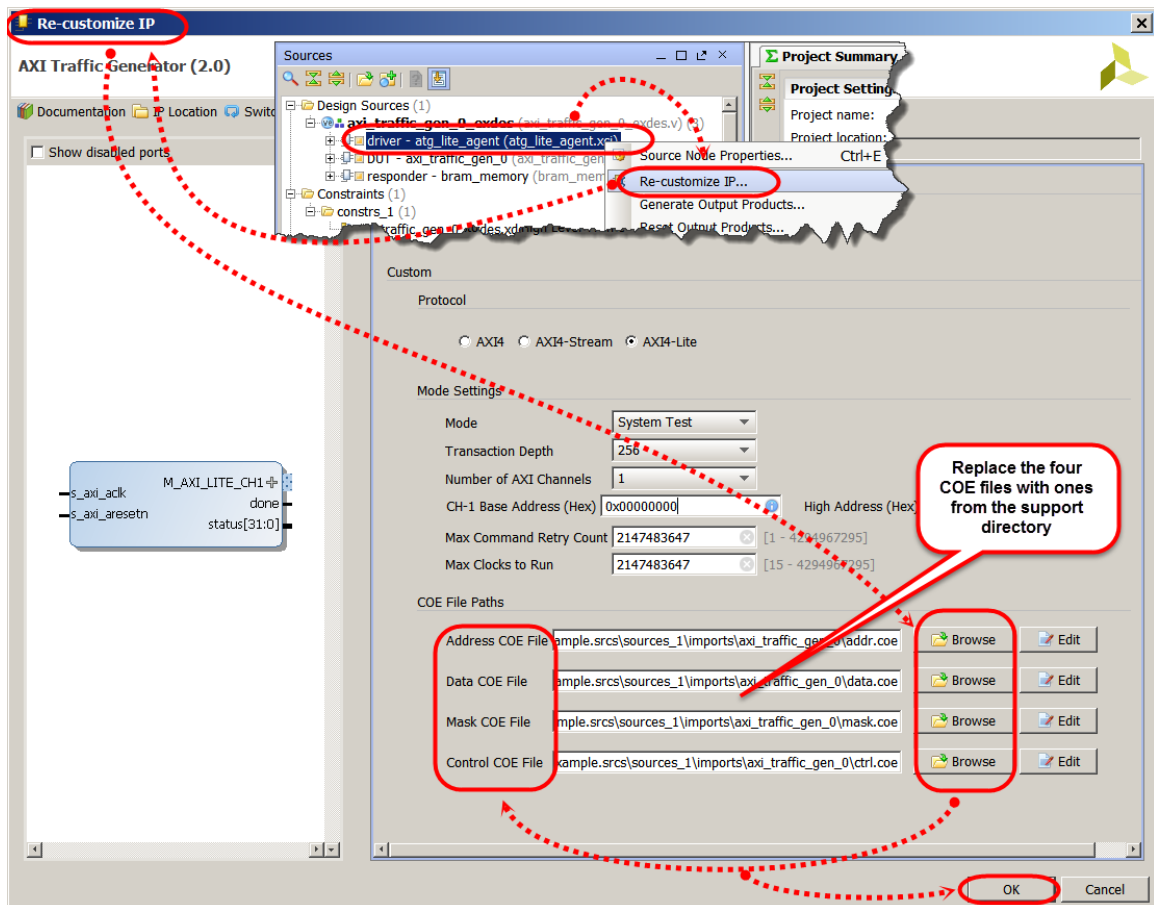


Figure 3-27: Replacing the COE Files for the Driver ATG

- 2-7-5. Accept the values and click **OK** to close the dialog box.
The Generate Output Products dialog box opens.

2-7-6. Click **Skip** because output products are not needed at this time.

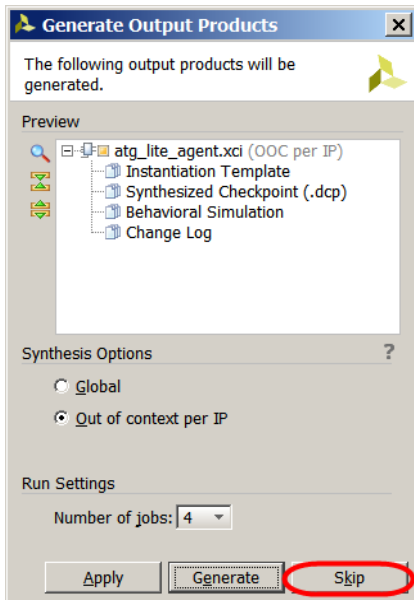


Figure 3-28: Skip Generating Output Products

2-7-7. Return to the Sources window.

2-7-8. Expand the **Coefficient Files** folder.

2-7-9. Double-click **addr.coe** to open the coefficients file in the editor.

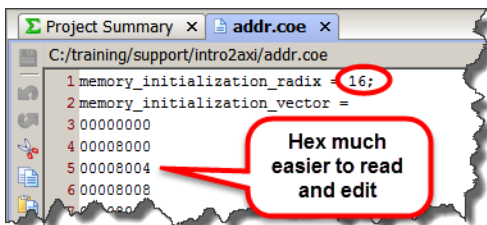


Figure 3-29: Hexadecimal COE File

2-7-10. Click the **X** next to the file name on the tab to close the file.

The following instruction 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.

2-8. Add simulation files to the design.

2-8-1. Select **Add Sources** under Project Manager in the Flow Navigator.

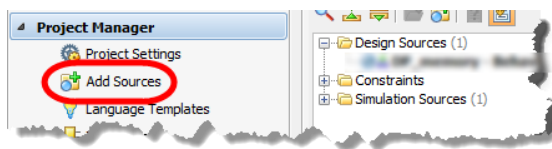


Figure 3-30: Selecting Add Sources

2-8-2. Select **Add or create simulation sources**.

2-8-3. Click **Next**.



Figure 3-31: Add Sources Dialog Box

2-8-4. Click the **Plus (+)** icon to open the pop-up menu.

2-8-5. Select **Add Files** to open the Add Source Files dialog box which allows you to browse to the desired directory.

2-8-6. Browse to the **C:\training\AXItransactions\support** directory if it is not open already.

2-8-7. Select **axi_traffic_gen_0_tb_top_behav.wcfg**.

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

2-8-9. Click **Finish** to add the file(s) to the project.

Simulating the Design and Analyzing the AXI Transactions Step 3

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, else the simulation will run with the old COE files. Since the COE files are only used for the Driver ATG component, only that one needs to reset. Note that 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 select it.

3-1-2. Select **Reset Output Products**.

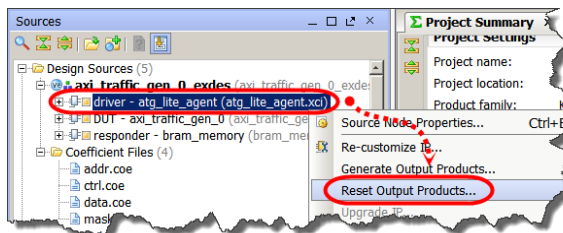


Figure 3-32: Reset Driver ATG Component Output Products

The Reset Output Products dialog box appears.

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

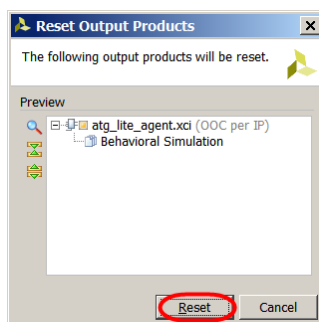



Figure 3-33: Confirming Output Products Reset

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

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

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

Note: If the simulator fails to start (reported problem in 2016.3), you can alternatively start the the Vivado simulator by entering `launch_simulation` in the Tcl Console.



- 3-2-2. After the waveform opens after approximately 1 to 2 minutes, click the **Run All** icon () to execute the entire simulation.

When done, the simulation will terminate at the last line of code executed.

- 3-2-3. Select the **axi_traffic_gen.xcfg** 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.

- 3-2-4. Click the **Zoom All** icon () to see the entire simulation.

- 3-2-5. Optional: Use the various controls to zoom and position the waveform cursor. Also, feel free to float the waveform window () and maximize to full screen (), and adjust the column width for maximum waveform exposure.

Note: If you are unfamiliar with the Vivado simulator, hovering the mouse pointer over an icon will describe its functionality.

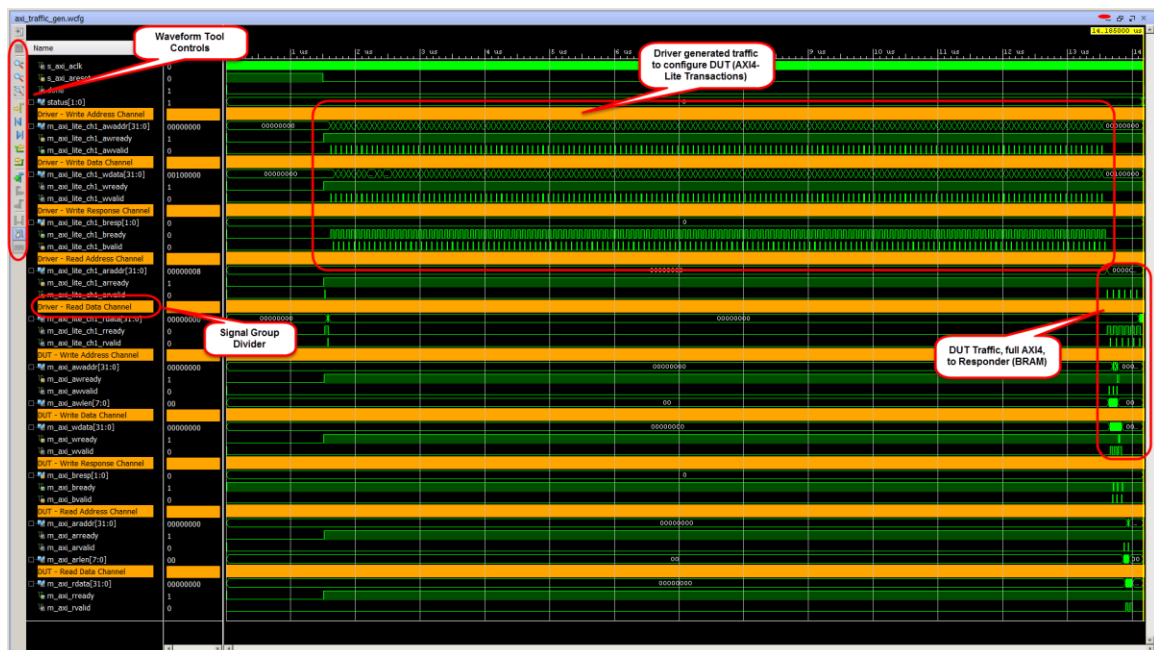


Figure 3-34: ATG Design Simulation Waveform

The testbench writes operational activities and messages to the Tcl Console. These will also aid in understanding the design.

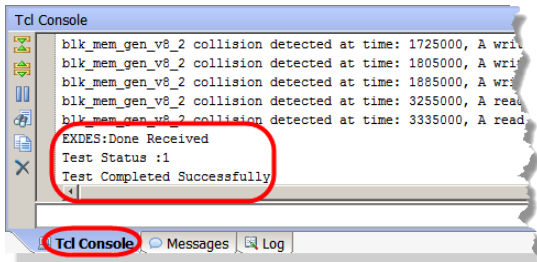


Figure 3-35: Simulation Console Messages

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 7

What do the waveform signal group dividers represent?

Question 8

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

Question 9

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 10

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

Question 11

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.

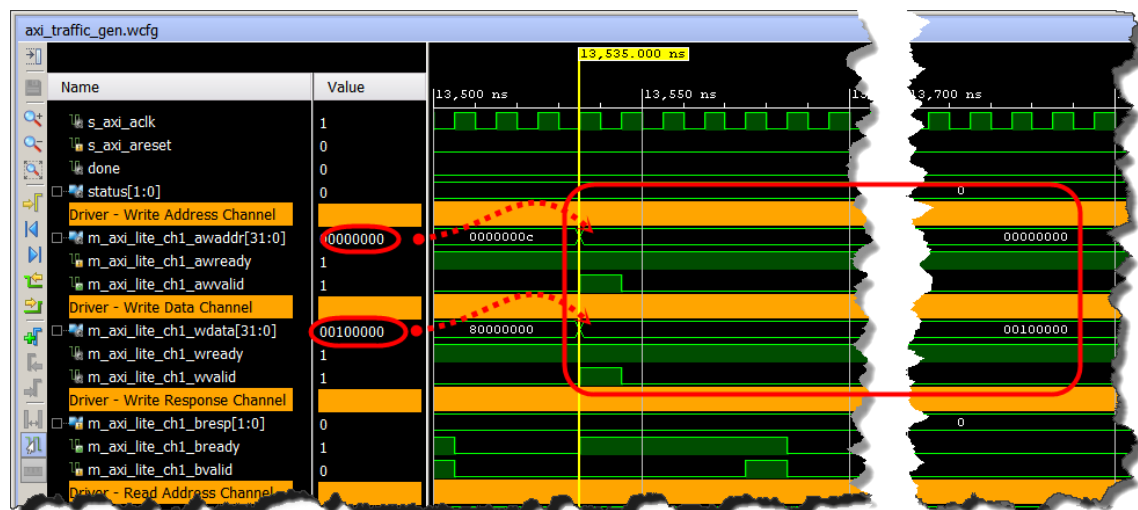


Figure 3-36: Enabling DUT – ATG to Begin Traffic Generation

Note that the rising edge **valid** signal is a great way to locate when a channels 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 is these different modes.



Figure 3-37: 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 12

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

These transactions make up the entire DUT traffic program that is to be generated. This was 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 also monitors these signals and ends the simulation with the appropriate console message.

Question 13

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]` (`arlen` 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 14

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 <code>awlen</code> or <code>arlen</code>
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 15

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.

This was just a demonstration of how to launch, use, and close the simulator.

- 3-4-1.** Select **File > Close Simulation** to exit from the simulator.
- 3-4-2.** Click **Yes** 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 address that the Driver ATG will write or read. 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.

Do note, as shown below, that 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.

Note that 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).

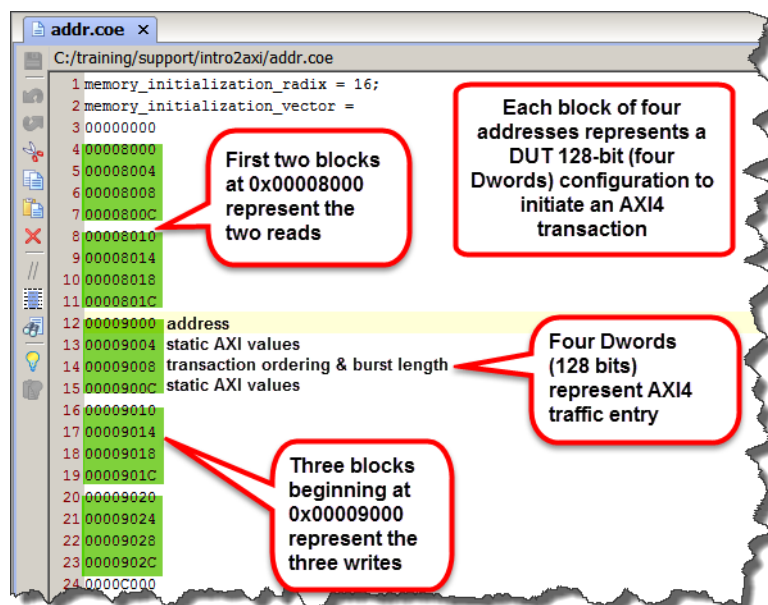


Figure 3-38: Driver Address COE File

3-5-3. From the Sources window, expand the **Coefficient Files** folder and double-click **data.coe** to open it.

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

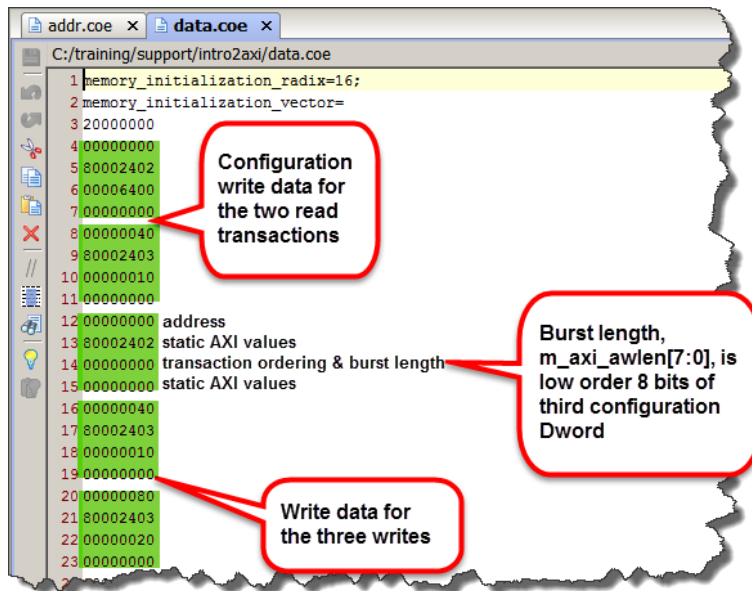


Figure 3-39: Unmodified Driver Data COE File

Question 16

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.)

3-5-4. Change the following lines in the *data.coe* file:

- Line 4: 0x00000040 – Address of first read transaction
- Line 8: 0x00000080 – Address of second read transaction
- Line 9: 0x80002404 – Burst length, *m_axi_arlen*, of second read transaction
- Line 13: 0x80002403 – Burst length, *m_axi_awlen*, of first write transaction
- Line 17: 0x80002401 – Burst length, *m_axi_awlen*, of second write transaction
- Line 21: 0x80002404 – Burst length, *m_axi_awlen*, of third write transaction

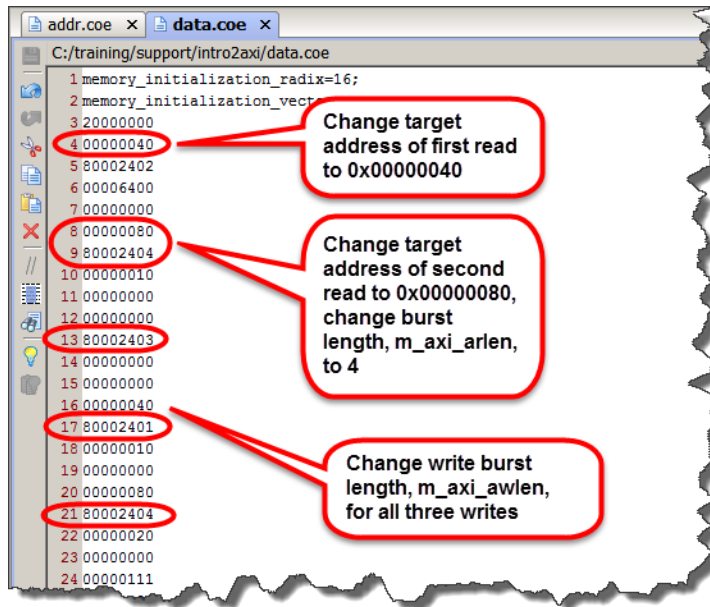


Figure 3-40: Modified Driver Data COE File

3-5-5. Select **File > 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 1 and 2 of this step.

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

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

Question 17

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.

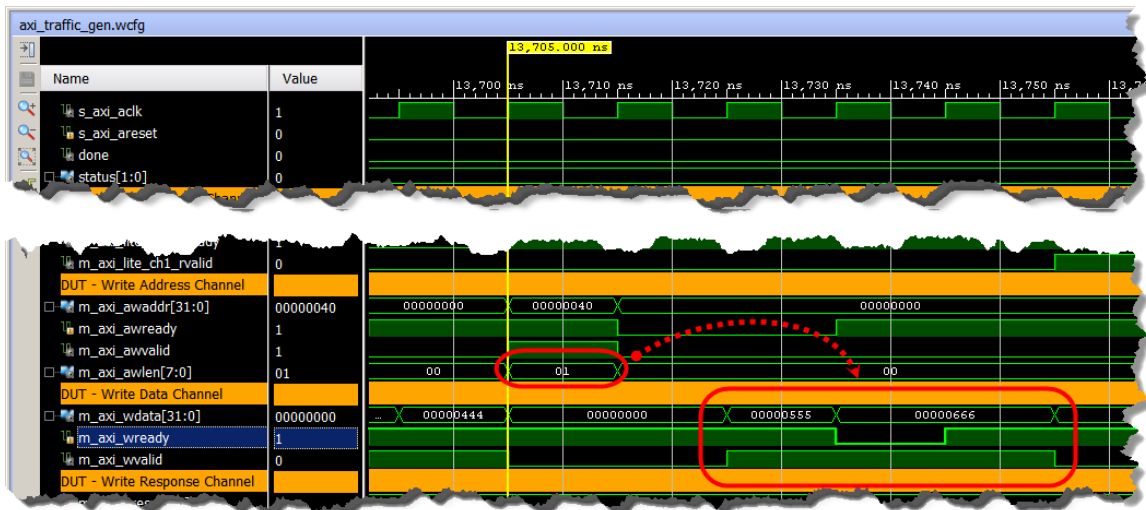


Figure 3-41: Second DUT – ATG Write

Question 18

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 19

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 20

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**.

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. Why was a board chosen (as opposed to an FPGA part), when the Vivado Design Suite project was created?

The ZC702 or ZedBoard 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. Which of the three components are really just 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.

7. 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.

8. 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.

9. 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 valid signal is generated by the AXI channel side (that is, providing the information) while the ready signal is generated by the receiving agent of the transaction.

10. 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).

11. 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.

12. 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.

13. 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.

14. 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 <code>awlen</code> or <code>arlen</code>
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.

15. 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.

16. 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.

17. 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

18. 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.

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

Yes.

20. 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.