

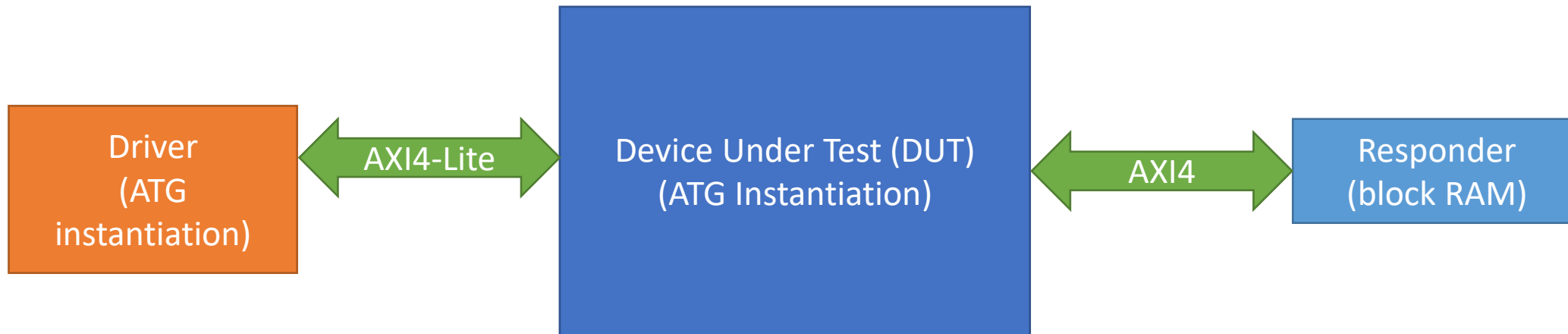
LAB04

AXI Traffic Generation (ATG)



What is a ATG?

- In this laboratory we will use two IP-cores ATG.
- First ATG is named driver and it is a custom profile Test Mode ATG.
- Second ATG is named DUT and it is a custom profile Advanced Mode ATG



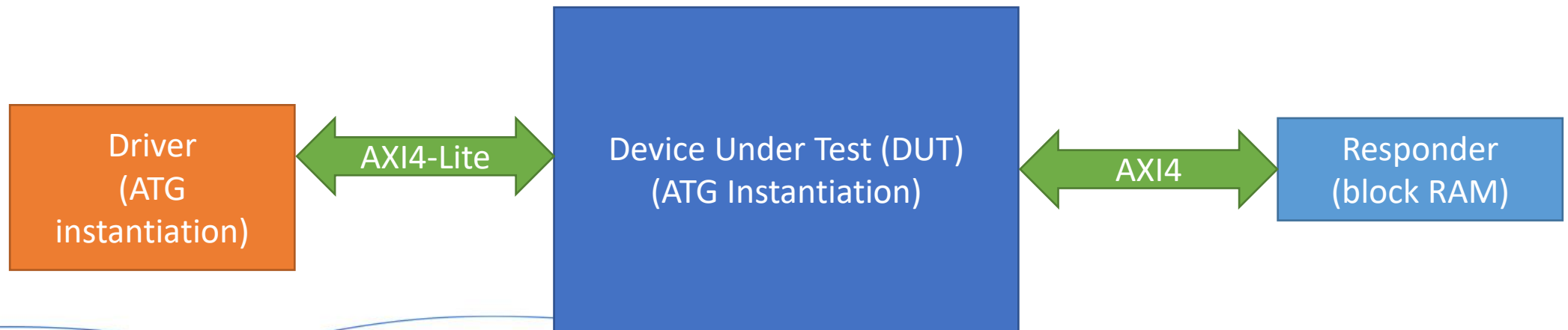
Driver: An instantiation of the ATG in AXI4-Lite mode. This module is used to generate AXI4 Lite transactions to program the DUT module.

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.

What is a ATG?

1. The driver will read the Master Register from DUT. (just for fun!)
2. The driver will fill the MSTRAM with data. (from COE FILES). This data Will be later used by the DUT.
3. The driver will fill the CMDRAM to set read and write the transactions that DUT shall implement with the block RAM. (from COE FILES)
4. The driver set the Master Register of DUT to start DUT operation.
5. The DUT perform the transactions sabs into its CMDRAM.
6. Operation is finished.



AXI Traffic Generator

ATG Customization

AXI Traffic Generator (2.0)

Documentation IP Location Switch to Defaults

☐ Show disabled ports

Component Name **axi_traffic_gen_0**

Profile Selection

☒ Custom ☐ High Level Traffic

Custom

Protocol

☒ AXI4 ☐ AXI4-Stream ☐ AXI4-Lite

Mode Advanced

Repeat Count 255 [2 - 16777215]

Write Address Gen Seed 0x7C9B [0 - 4294967295]

Read Address Gen Seed 0x5A5A [0 - 4294967295]

Address Width 32 [32 - 64]

Slave Interface

Data Width 32 [32 - 1024]

ID Width 1 [0 - 32]

AWUSER Width 8 [0 - 32]

ARUSER Width 8 [0 - 32]

Base Address (Hex) 0x00000000 [0 - 4294967295]

High Address (Hex) 0x0000FFFF [0 - 4294967295]

Master Interface

Data Width 32 [32 - 1024]

Thread ID Width 1 [0 - 6]

AWUSER Width 8 [0 - 16]

External start control

Traffic data from internal block RAM

Generate full AXI4 traffic on master interface

AXI4 traffic generation options (more are in internal block RAM)

ATG internal block RAM and registers accessed over slave port

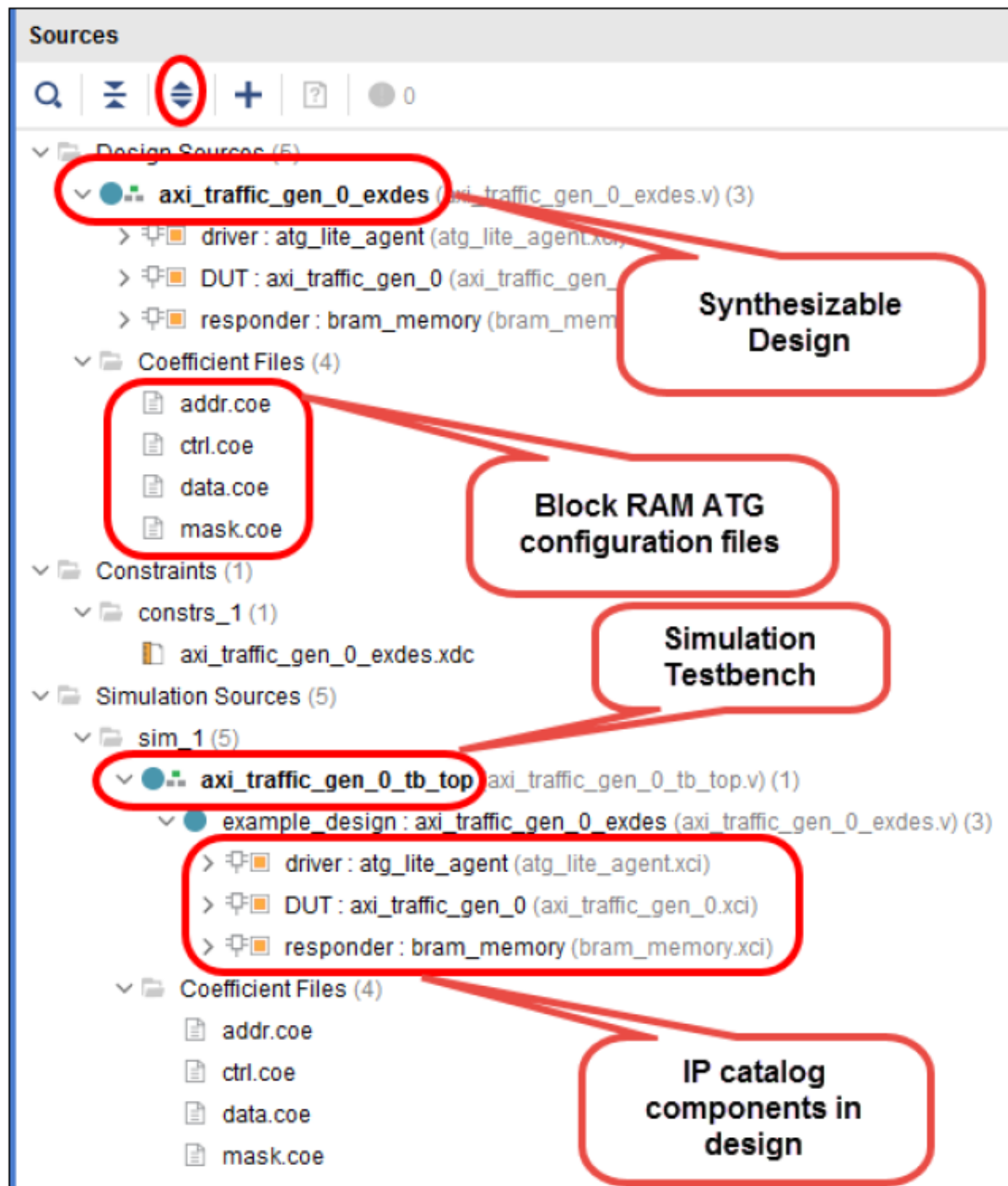
Base address space of internal block RAM and registers

Output port of generated AXI4 traffic

OK Cancel

AXI Traffic Generator

Example: Driver-DUT-RAM

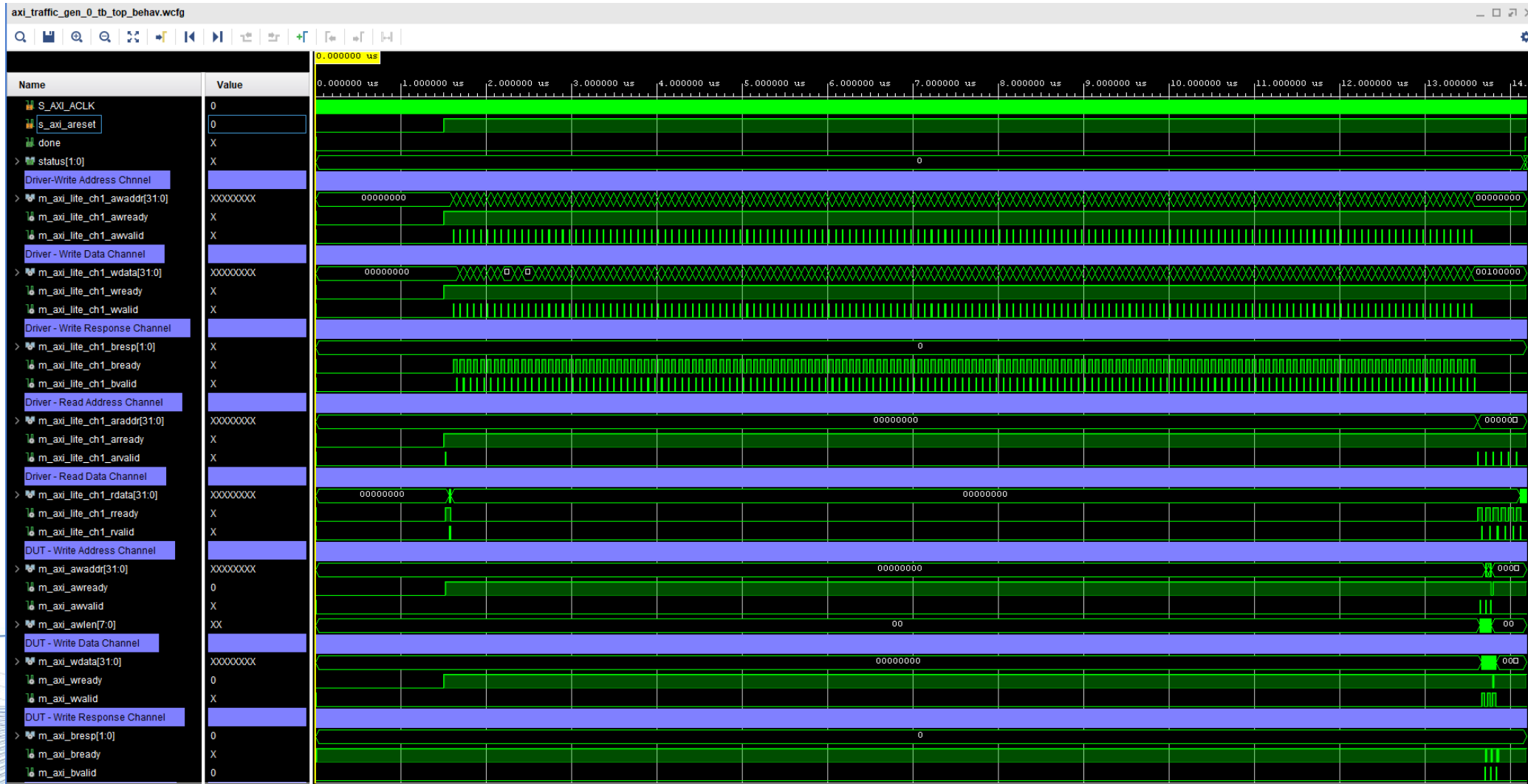


The screenshot displays the 'Sources' window of the Xilinx IDE, showing the project structure for 'axi_traffic_gen_0_exdes'. The project is organized into several categories:

- Design Sources (5)**: Contains the main design files.
 - axi_traffic_gen_0_exdes** (axi_traffic_gen_0_exdes.v) (3): The main design file, highlighted with a red circle and a callout box labeled **Synthesizable Design**.
 - driver : atg_lite_agent** (atg_lite_agent.xci): The driver component.
 - DUT : axi_traffic_gen_0** (axi_traffic_gen_0.xci): The Design Under Test (DUT) component.
 - responder : bram_memory** (bram_memory.xci): The responder component.
- Coefficient Files (4)**: Contains configuration files for the Block RAM ATG.
 - addr.coe
 - ctrl.coe
 - data.coe
 - mask.coeA callout box labeled **Block RAM ATG configuration files** points to this group.
- Constraints (1)**: Contains the constraints file.
 - constrs_1** (1): Contains the file **axi_traffic_gen_0_exdes.xdc**. A callout box labeled **Simulation Testbench** points to this file.
- Simulation Sources (5)**: Contains the simulation testbench and its components.
 - sim_1** (5): Contains the simulation testbench file **axi_traffic_gen_0_tb_top** (axi_traffic_gen_0_tb_top.v) (1), highlighted with a red circle and a callout box labeled **Simulation Testbench**.
 - example_design : axi_traffic_gen_0_exdes** (axi_traffic_gen_0_exdes.v) (3): The main design file, highlighted with a red circle and a callout box labeled **IP catalog components in design**.
 - driver : atg_lite_agent** (atg_lite_agent.xci): The driver component.
 - DUT : axi_traffic_gen_0** (axi_traffic_gen_0.xci): The Design Under Test (DUT) component.
 - responder : bram_memory** (bram_memory.xci): The responder component.
- Coefficient Files (4)**: Contains configuration files for the Block RAM ATG.
 - addr.coe
 - ctrl.coe
 - data.coe
 - mask.coe

AXI Traffic Generator Simulation

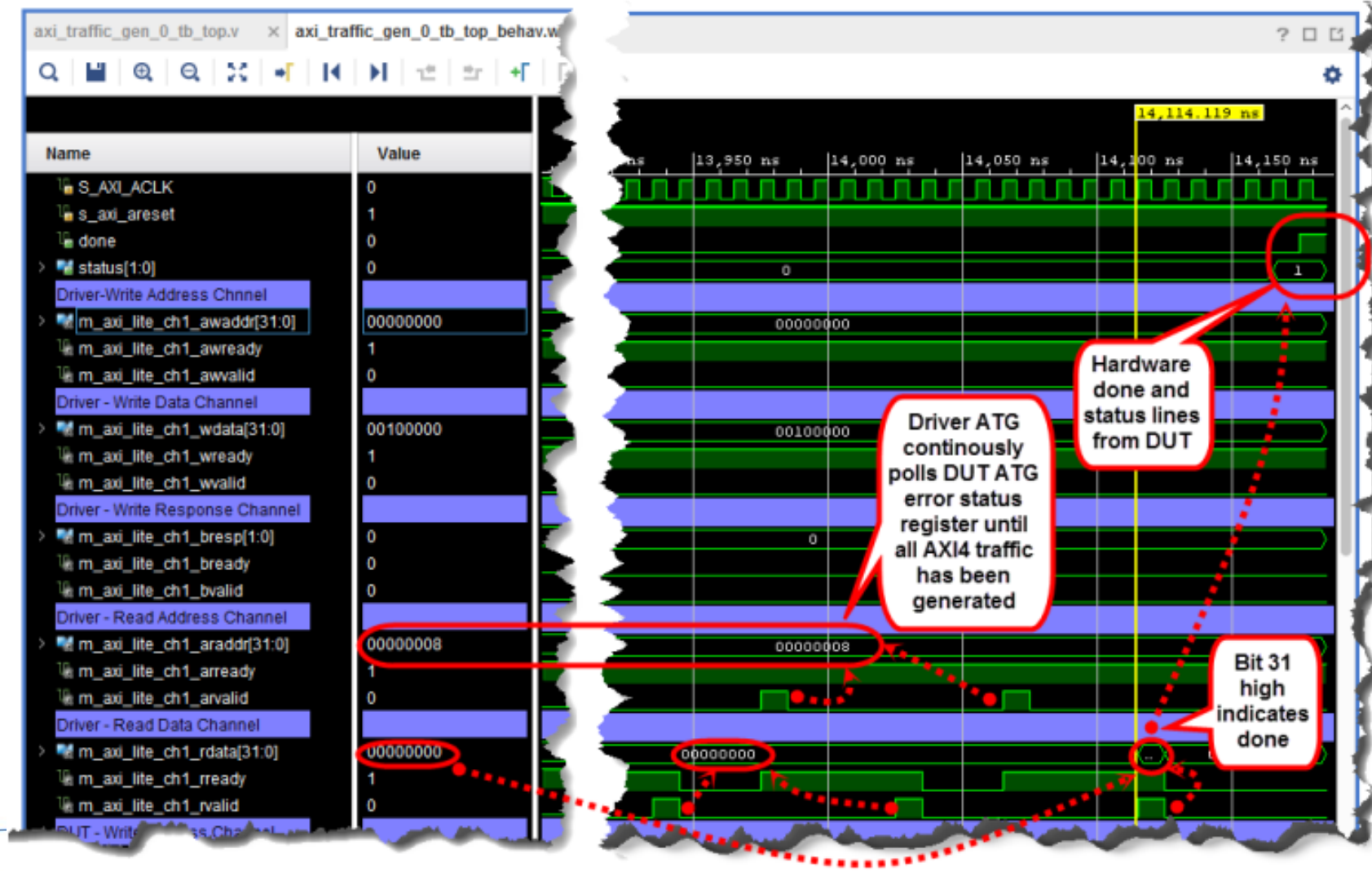
After Simulating for 15us you should see something like this:



Do the following (Driver):

1. How many AXI ports can you identify? Which are their channels?
2. Check all names of the signals and match them with theory. Which two signals are present in each channel? (handshaking signals)
3. Zoom in around time 1.5us to 1.7us:
 1. Which is the first transaction of the driver? How do you identify it?
 2. Which is the second transaction of the driver? How do you identify it?

Note: To start DUT transaction the bit 20 of the Master Control Register (0x00000000) must be set to 1. (ATG [guideline](#)). Find where this happens.
4. Go to time 13us to 14us and identify the end of writing transaction from driver to LUT.



Do the following (DUT):

1. Go to time 13.65us identify how many transactions make the LUT.
2. Which difference can you identify compared to AXI Lite transactions.

Note: Awlen[7:0]: defines the number of bursts

Using awreadt, wvalid, rready and rvalid determine the number of beats per transaction.

3. Complete the following table for all DUT transactions

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

Do the following (DUT):

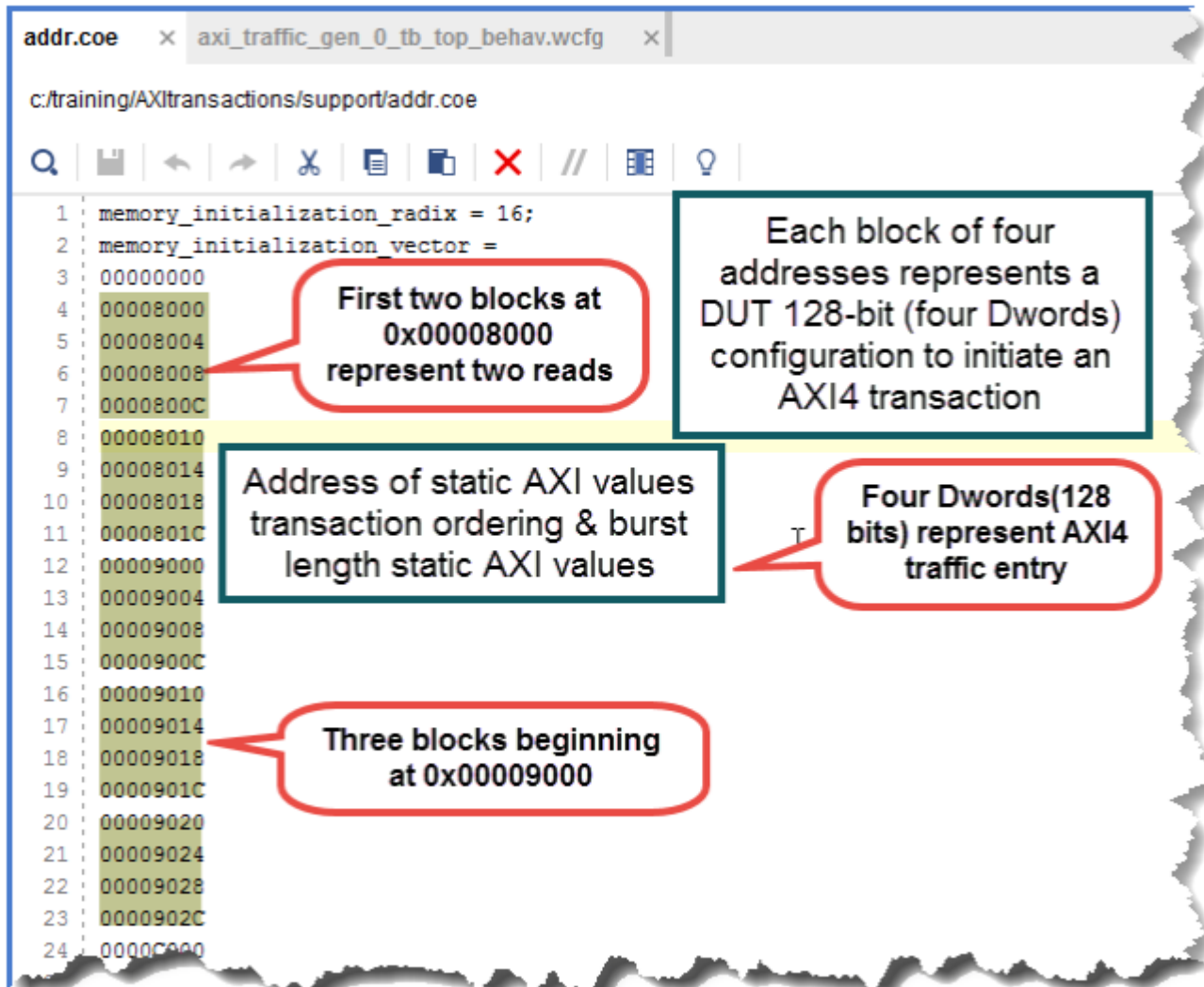
1. Go to time 13.65us identify how many transactions make the LUT.
2. Which difference can you identify compared to AXI Lite transactions.

Note: Awlen[7:0]: defines the number of bursts

Using awreadt, wvalid, rready and rvalid determine the number of beats per transaction.

3. Complete the following table for all DUT transactions

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

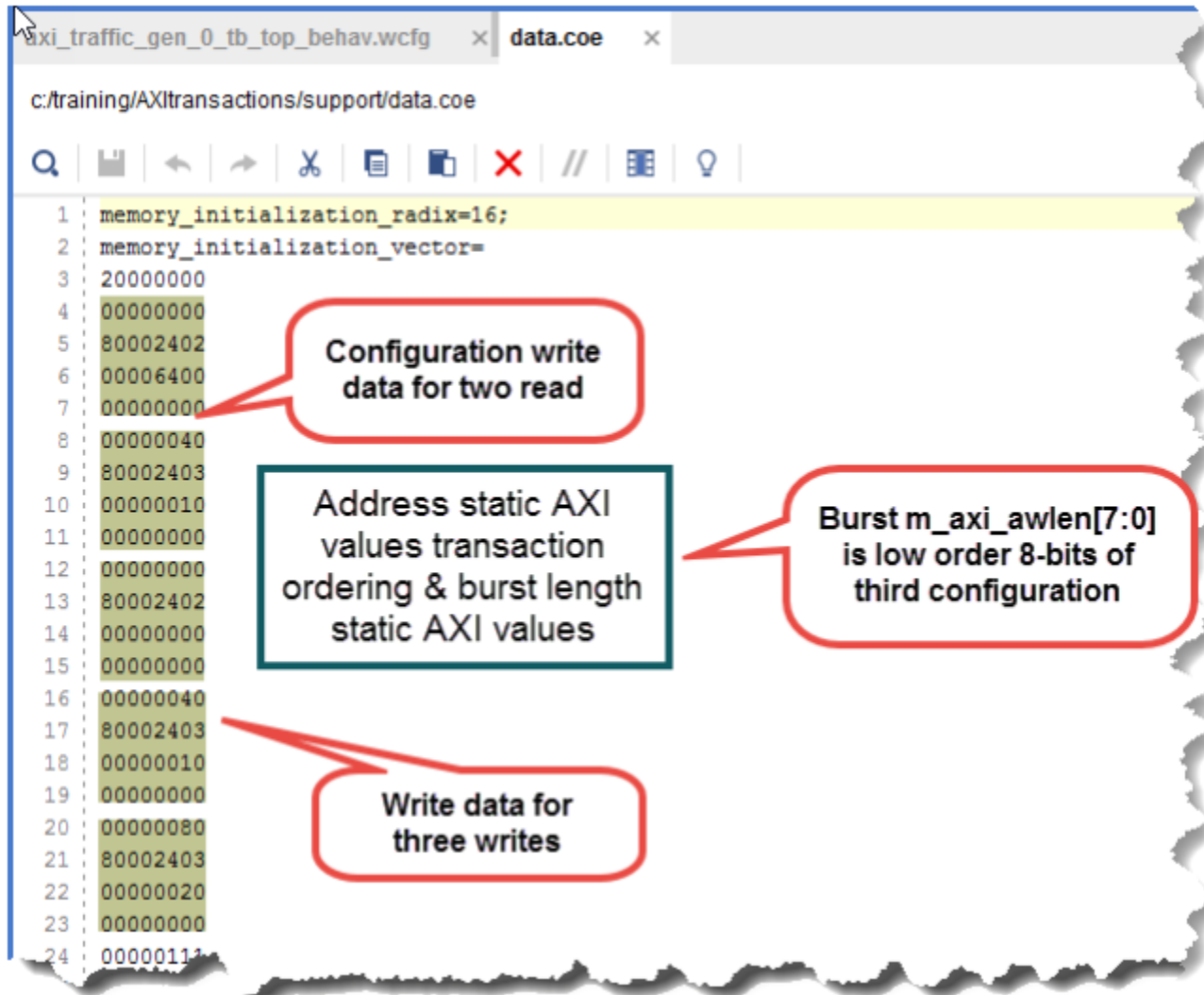


The screenshot shows a text editor with the file `addr.coe` open. The code defines a memory initialization vector for an AXI4 transaction. Annotations explain the structure of the data:

- First two blocks at 0x00008000 represent two reads**: Points to the first two entries in the vector (lines 4-5).
- Each block of four addresses represents a DUT 128-bit (four Dwords) configuration to initiate an AXI4 transaction**: A general note about the block structure.
- Address of static AXI values transaction ordering & burst length static AXI values**: Points to the third entry (line 10).
- Four Dwords(128 bits) represent AXI4 traffic entry**: Points to the third entry (line 10).
- Three blocks beginning at 0x00009000**: Points to the last three entries in the vector (lines 16-18).

```
1 memory_initialization_radix = 16;
2 memory_initialization_vector =
3 00000000
4 00008000
5 00008004
6 00008008
7 0000800C
8 00008010
9 00008014
10 00008018
11 0000801C
12 00009000
13 00009004
14 00009008
15 0000900C
16 00009010
17 00009014
18 00009018
19 0000901C
20 00009020
21 00009024
22 00009028
23 0000902C
24 0000C000
```

1. The address COE file contains a list of addresses that the Driver reads and writes.
2. Read and write operations are coded into `crt.coe`
3. The data written or read (predicted) is contained in `data.coe`
4. The DUT is configured to make 5 transactions.
5. Each DUT transaction require 128bits total (all AXI data/config). It is divided into 4 data of 32bits, name dword.
6. Within DUT ATG there are two identical structures, one for reading (starting at 0x00008000). And for writing, beginning at address 0x00009000.



The image shows a screenshot of a text editor window with the file path `c:/training/AXItransactions/support/data.coe`. The editor contains a COE file with the following content:

```
1 memory_initialization_radix=16;  
2 memory_initialization_vector=  
3 20000000  
4 00000000  
5 80002402  
6 00006400  
7 00000000  
8 00000040  
9 80002403  
10 00000010  
11 00000000  
12 00000000  
13 80002402  
14 00000000  
15 00000000  
16 00000040  
17 80002403  
18 00000010  
19 00000000  
20 00000080  
21 80002403  
22 00000020  
23 00000000  
24 00000111
```

Annotations are present in the image:

- A red callout bubble points to lines 5 and 6, containing the text: "Configuration write data for two read".
- A blue-bordered box highlights lines 10 through 15, containing the text: "Address static AXI values transaction ordering & burst length static AXI values".
- A red callout bubble points to line 11, containing the text: "Burst m_axi_awlen[7:0] is low order 8-bits of third configuration".
- A red callout bubble points to lines 19 and 20, containing the text: "Write data for three writes".

1. Each entry corresponds to the data associated to the same line on the address coe file.
2. When we write to adress 0x8000 and 0x9000 we must write 128bit data. That is why we have blocks of four data (12bits each).
3. First two blocks is the data that configure the two reading transactions.
4. Blocks 3,4,and 5 configure three reading transactions.

1. Modify the COE files to change the address read and written by the DUT.
2. Modify the COE files to change the length of the read and written burst.
3. Modify the COE files to change the data written to the RAM by the DUT.

Remember that changing the COE files you only change the data that the driver is sending to the DUT. However, this data is strategically sent to configure the transactions of the DUT.

Now you can add AXI drivers (with predefined data) to connect them with your own blocks that possess AXI-lite communication protocol.

You can also configure an AXI4-full interface to connect it to any of your own blocks designed as AXI4-full slaves.

AXI Traffic Generator Overview





Electrical Engineering Department
Pontificia Universidad Católica de Chile
peclab.ing.uc.cl