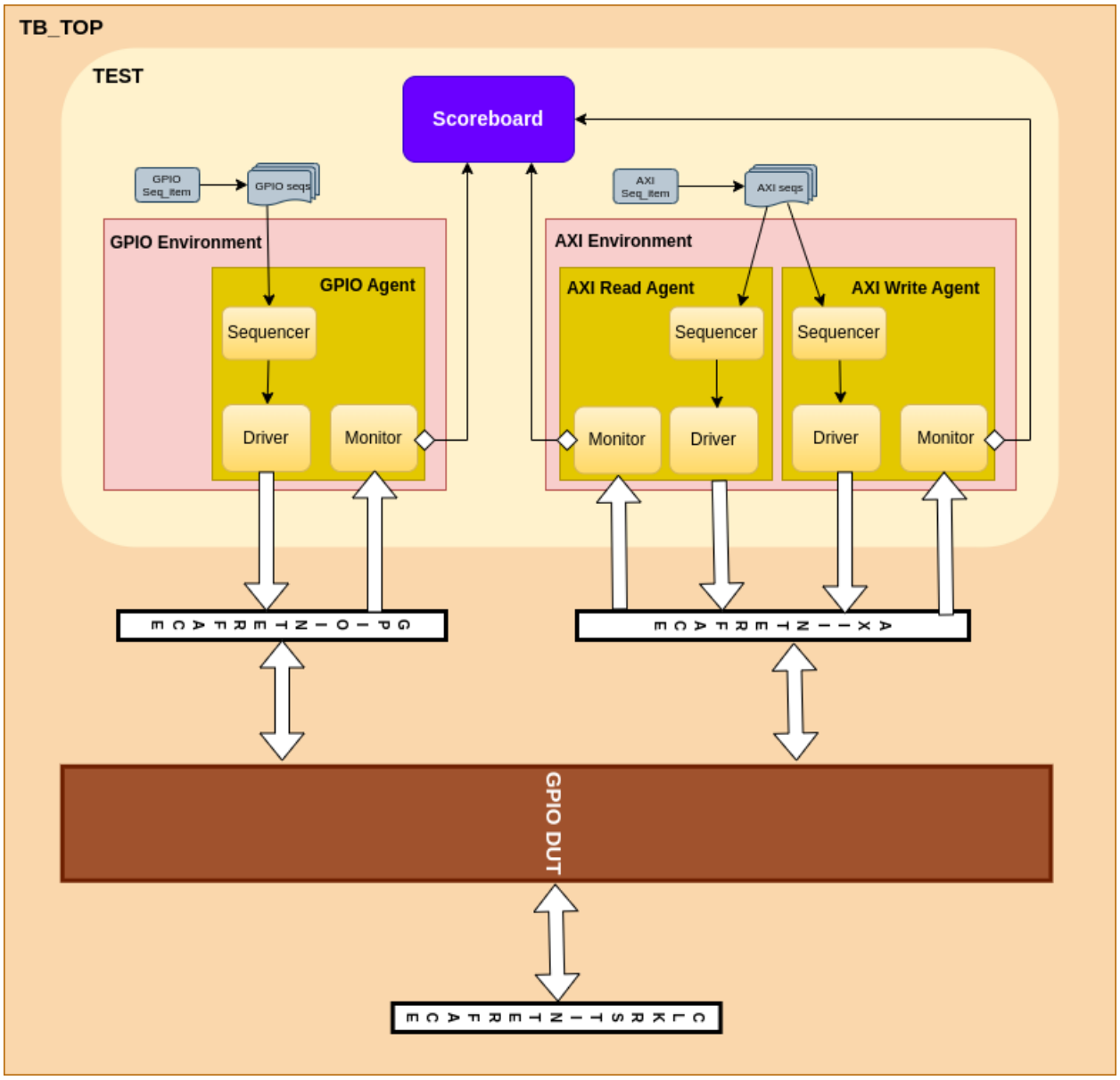
TCP – AXI-GPIO Verification

TCP overview:

The goal of this project is to create a UVM based verification environment for **Xilinx LogiCORE IP AXI-GPIO** core, which provides a general purpose input/output interface to **AXI4-Lite interface**. Two environments are created, one for AXI4-Lite interface and other for GPIO interface. The two of them are integrated together in the base test. Different test cases are designed in which sequences for both environments are run to achieve the desired result for that test case.

Verification Testbench Architecture:



TestBench Top:

All the relevant files for the verification are included as header in the top module. Following parameters are set for this design:

C_S_AXI_ADDR_WIDTH = 9
C_S_AXI_DATA_WIDTH = 32
C_GPIO_WIDTH = 32
C_GPIO2_WIDTH = 32

Clock and reset are declared. 3 static interfaces **axi_intf**, **gpio_intf** and **clk_rst_if** are created and set using config_db. The design module **"axi_gpio_0"** is instantiated and the DUT signals are connected with the corresponding interface signals. The clock is set active through the clk_rst_if interface. Finally the specific test is set to run.

Tests Library:

Base_test:

The **axi_env**, **gpio_env** and **scoreboard** are instantiated here. All 3 interfaces are set here using config_db. The respective monitors' analysis ports are connected to the respective ports in the scoreboard in the connect phase. **Reset** is applied in the reset phase. Moreover, an objection is raised in the pre main phase and dropped in the post main phase of the run phase.

The objection is also raised in the main phase of the run phase of the extended test classes. After running the required sequences the objection is dropped in the same phase. The relevant sequences are instantiated in the respective test classes extended from this base_test.

Topology Report is printed at the end of the elaboration phase to observe the hierarchy of the testbench. The detailed explanation of all instantiated components is as follows:

Interfaces:

Following interfaces are used for communication between DUT and the testbench:

- **axi_intf:** monitors AXI write address, write data, response, read address, read data response signals. This acts as the AXI slave interface. The clocking blocks for corresponding driver and monitor are created.
- **gpio_intf:** monitors GPIO input, output and direction signals for both channels. Also monitors the interrupt signal if enabled. The clocking blocks for corresponding driver and monitor are created.

- **clk_rst_if:** monitors clock and reset. It is mainly responsible for activating the clock and resetting the design.

AXI Environment:

Two agents are built in this environment:

Write_Agent:

This agent controls 3 channels of AXI i.e write address, write data and response channel.

- **Write_Sequencer:** Drives the write sequences.
- **Write_Driver:** Gets the next item from write_sequencer and drives the signals onto the axi_intf following the AXI protocol. The 3 channels are driven in parallel.
- **Write_Monitor:** Waits for the output condition for each channel, gets the signals from the axi_intf and then writes them to the analysis port. The 3 channels are monitored in parallel.

These handles of these 3 classes are instantiated in agent. The seq_item_export of the sequencer is connected to the seq_item_port of the driver.

Read_Agent:

This agent controls the remaining 2 channels of AXI i.e read address and read data response channel.

- **Read_Sequencer:** Drives the read sequences.
- **Read_Driver:** Gets the next item from read_sequencer and drives the signals onto the axi_intf following the AXI protocol. The 2 channels are driven in parallel.
- **Read_Monitor:** Waits for the output condition for each channel, gets the signals from the axi_intf and then writes them to the analysis port. The 2 channels are monitored in parallel.

These handles of these 3 classes are instantiated in agent. The seq_item_export of the sequencer is connected to the seq_item_port of the driver.

GPIO Environment:

One agent for GPIO is instantiated in this environment.

GPIO_Agent:

This agent controls the signals of the gpio_intf.

- **GPIO_Sequencer:** Drives the GPIO sequences.
- **GPIO_Driver:** Gets the next item from gpio_sequencer and drives the signals onto the gpio_intf following the GPIO protocol. The 2 channels are driven sequentially.

- **GPIO_Monitor:** Gets the signals from the gpio_intf, compares them with the previous_gpio_intf signal. If there is any transition in one of them, then the signals are written to the analysis port and the previous_gpio_intf signals are updated with new ones.

These handles of these 3 classes are instantiated in agent. The seq_item_export of the sequencer is connected to the seq_item_port of the driver.

ScoreBoard:

In scoreboard, 3 analysis ports are declared: 1 gets the **gpio_pkt** from gpio_monitor analysis port, and the other 2 gets the **axi_pkt** from analysis ports of AXI write_monitor and read_monitor. Some incoming axi_pkts are compared with the corresponding gpio_pkts to verify them correctly. These usually include the packets that set the direction of GPIO pins and write or read from GPIO pins. Other axi_pkts are verified by checking whether the valid value is written or not. These include writing the global and local interrupt registers. The **total incoming packets**, **matched packets** and **mismatched packets** are counted and then reported in the report phase.

Axi_Seq_Item:

A transaction is created for the **axi_intf** signals. All signals are declared as logic. The write address, write data and read address signals are declared as rand.

Gpio_Seq_item:

A transaction is created for the **gpio_intf** signals. All signals are declared as logic. The channel 1 & 2 input signals are declared as rand.

Gpio_Seqs:

A **gpio_base_seq** creates the handle for gpio_seq_item. From this, a simple **gpio_seq** is extended that generates some random values for channel 1 & 2 inputs.

Axi_Seqs:

An **axi_base_seq** creates the handle for axi_seq_item. From this, two separate sequences for read and write are extended:

Axi_write_seq:

A simple axi_write_seq creates the sequence that randomizes the write address and write data signals. From this, some fixed sequences are extended which are then used in the specific test accordingly. Those sequences are as follows:

- Axi_write_tir_1_in
- Axi_write_tir_1_out
- Axi_write_data_1

- Axi_write_tir_2_in
- Axi_write_tir_2_out
- Axi_write_data_2
- Axi_write_GIER
- Axi_write_IPIER_1
- Axi_write_IPIER_2
- Axi_write_IPIER_1_2
- Axi_write_IPISR_1
- Axi_write_IPISR_2
- Axi_write_IPISR_1_2

Axi_read_seq:
A simple axi_read_seq creates the sequence that randomizes the read address signal. From this, some fixed sequences are extended which are then used in the specific test accordingly. Those sequences are as follows:

- Axi_read_data_1
- Axi_read_tir_1
- Axi_read_data_2
- Axi_read_tir_2
- Axi_read_GIER
- Axi_read_IPIER
- Axi_read_IPISR

Test cases:

Following test cases were made for the testing of the AXI-GPIO verification environment. All the tests were PASSED.

- GPIO_ch_1_all_input
- GPIO_ch_1_all_output
- GPIO_ch_2_all_input
- GPIO_ch_2_all_output
- GPIO_ch_1_2_input
- GPIO_ch_1_2_output
- GPIO_ch_1_input_2_output
- GPIO_ch_1_output_2_input
- GPIO_ch_1_2_intr_en_with_input_at_ch_any
- GPIO_ch_1_intr_en_with_input_at_ch_any
- GPIO_ch_2_intr_en_with_input_at_ch_any